2005

Solving the timetabling problem using constraint satisfaction programming

Lixi Zhang

University of Wollongong

Solving the timetabling problem using constraint satisfaction programming

A thesis submitted in partial fulfilment of the requirements for the award of the degree

Master of Information System by Research

From

University of Wollongong

By

Lixi ZHANG

Information Systems

School of Economics and Information Systems

2005

Thesis Certification

I, Lixi Zhang, declare that this thesis, submitted in partial fulfilment of the requirements for the award of the Degree of Master of Information Systems by Research, in the Information Systems discipline, School of Economics and Information Systems at the University of Wollongong, is wholly my own work otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Lixi Zhang

31 March 2005

Table of Contents

List of Publication

This is a refereed conference paper related to this research.

Zhang, L. X. and Lau, S. K. 2004, "Solving the timetabling problem using constraint satisfaction programming", *Proceedings of the 2004 International Conference on Computational Intelligence for Modelling, Control and Automation*, 12-14 July, 2004, Gold Coast, Australia, pp.291-298.

List of Figures

Page No.

List of Tables

Abstract

The timetabling problem is well known and much research has been done about it. It consists of dealing with a number of different factors, such as the number of lectures, subjects, classrooms, working time slots, and various other constraints. The problem consists of a set of subjects to be scheduled in timeslots, a set of rooms in which time can take place, a set of students who attend the subjects, and a set of subjects satisfied by rooms and required by timeslots. The heart of the problem is the constraints that exist as regulations within each resource and between resources. The problem exhibits the unwelcome nature of combinatorial explosion.

There are various solution approaches to solve the timetabling problem. This research applies constraint satisfaction programming approach to address the problem. Constraint satisfaction programming is a paradigm in Artificial Intelligence, which has been used successfully to solve many scheduling problems. Constraint satisfaction problem (CSP) is defined by a set of variables, a domain of values for each variable, and a set of constraints between each pair of variables. A solution of a CSP is a consistent assignment of all variables to values in such a way that all the constraints are satisfied. Backtracking with arc consistency is the basic of the search algorithm for applying constraint propagation. Strategies of variable and value ordering play an important role in solution efficiency.

This research focuses on developing a CSP model for a university timetabling problem. A sample case study problem is investigated and a constraint satisfaction programming approach is implemented using ILOG Scheduler and ILOG Solver. We have used various goals in ILOG to investigate the performance of the CSP approach. Our results have shown that enforcing tight constraint level and ranking the constraints by positioning the constraints at the beginning of the activities can lead to better result.

Acknowledgement

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I am deeply indebted to my supervisor Dr. Sim Kim Lau whose help, stimulating suggestions, invaluable assistance and encouragement helped me in all the time of research for and writing of this thesis. My colleagues from the Decision Systems Laboratory supported me in my research work. I want to thank them for all their help, support, interest and valuable hints. Especially I am obliged to Mr. Seung Hwan Kang, Ms. Qiuming Lin and Mr. Peter Harvey. I would like to thank the Department of Information Systems for financially supporting me to attend the Conference.

I would like to express thanks to my inspirers----friends, colleagues, and foremost my family. Their patience made me accepting the idea of sharing experiences and insights with you. Especially, I would like to give my special thanks to my husband whose patient love enabled me to complete this work.

Chapter 1 Introduction

The timetabling problem is a historic problem. It generally is considered as a resources allocation problem with permutation and combination characteristics. The one unique feature of the timetabling problem is the constraints are of both hard and soft nature. Hard constraints are functional constraints that all feasible solutions must satisfy, and soft constraints are the preference type of requests. With the understanding that it is impossible to satisfy all constraints, one needs to look for a solution approach that can satisfy all hard constraints and as many soft constraints as possible.

This research proposes a solution for a university timetabling problem using the constraint satisfaction programming approach. This chapter is organized as follows. Section 1 discusses the research problem. Section 2 presents research aims and objectives. Section 3 gives an overview of how this thesis is organised.

1.1 Research problem

Every year or term in a university, every individual department has to design a new timetable for subjects. The timetabling problem consists of placing these subjects, which share resources, such as lecturers and classrooms, in a weekly calendar. Solutions to timetabling problems have been proposed since the 1960s (Almond, 1966; Brittan and Farley, 1971; Vit'anyi, 1981; Tripathy, 1984; de

Werra, 1985; Costa, 1994; Abramson, 1991; Abramson and Abela, 1991; Hertz, 1992; Burke et al., 1994; Jaffar and Maher, 1994; Gunadhi et al., 1996; Guéret et al., 1996; Lajos, 1996; Deris et al., 1997; Terashima-Marín, 1998; Schaerf, 1999a; Schaerf, 1999b; Brailsford et al., 1999; Abdennadher and Marte, 2000). Different timetabling problems have different constraints. This can make them very difficult to solve. A common constraint is 'no student is assigned to two or more subjects simultaneously' which is known as a hard constraint because it should, under no circumstances be violated. Other constraints known as soft constraints are desirable but it is not essential to satisfy them. Indeed, it would usually be impossible to satisfy all of them in a given problem.

This research will attempt to solve a timetabling problem for a university department using the constraint satisfaction programming (CSP) approach.

1.2 Research Aim and Objectives

This research investigates the use of the CSP approach to solve a university timetabling problem. The objectives of this research are:

1. To study the feasibility of solving university timetabling problem using CSP approach.
2. To develop a CSP model for the timetabling problem.
3. To solve the timetabling problem using ILOG Scheduler and ILOG Solver.

4. To investigate the performance of the CSP approach by comparing different goals using ILOG Solver.

1.3 Organization of Thesis

The thesis is organized as follows:

- Chapter 2 is the literature review. It provides background knowledge of timetabling problem and the approaches used to solve the problem. We have given a definition of the CSP and arc-consistency. We have also included discussion of backtracking, forward checking, look ahead maintaining arc consistency, back-jumping and back-marking search algorithms to solve the CSP.
- Chapter 3 presents the sample case study for the timetabling problem. In this chapter we have discussed the implementation of the CSP approach using the ILOG Scheduler and Solver. Background information on ILOG Scheduler and Solver will be presented.
- Chapter 4 presents the results of the sample case study problem. We have considered various scenarios of solving the problem in ILOG include: ranking goals, SetTime goals and enforcement levels.
- Chapter 5 is the conclusion. We have also proposed future research direction.

Chapter 2 Literature Review

2.1 Introduction

Solving a real-world timetabling problem manually often requires a significant amount of computation time. A lot of researches have been invested to provide automated support for human timetables. The solution approaches to timetabling problem can be broadly classified into the following categories: i) Operations Research (OR), which range from the use of mathematical programming to heuristics (e.g. graph colouring and network flow techniques); ii) Human-Machine Interaction, where an initial feasible solution is obtained by the machine, and subsequently improved upon manually; and iii) Artificial Intelligence (AI), which involve tools such as Expert Systems (ES) and Artificial Neural Networks (ANN) to help get the feasible solutions (Gunadhi et al., 1996). Other AI algorithms used include simulated annealing, tabu search, genetic algorithms and constraint satisfaction approach.

This chapter presents literature review on approaches and methods used to solve the timetabling problem. The rest of the chapter is organized as follows. Section 2 discusses the characteristics of timetabling problem. Section 3 presents application of OR method in solving the timetabling problem. Human-machine

interaction approach is discussed in Section 4. Section 5 presents the AI approaches. Section 6 discusses the constraint satisfaction programming approach. We will give the definition of the CSP and discuss the use of backtracking, look ahead schemes (forward checking and look ahead maintaining arc consistency) and look back schema (back-jumping and back-marking) to solve CSP. Finally section 7 concludes the chapter.

2.2 The Timetabling Problem

Anyone who has participated in an educational system that allows choices in student courses has experienced the problem of course or subject scheduling. Intuitively, the problem is easy to understand: students want to take certain courses and lecturers need to teach those courses at times when the students can attend. Unfortunately, the problem is extremely difficult to solve. All problems related to building a timetable are known to be NP-Complete (Duncan, 1965; Even et al., 1976; Hertz, 1992).

Wren (1996, p.46) defines the timetabling problem as a special case of scheduling**:** **"***Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives*." In a more common definition, timetabling problem is the assignment of time slots to a set of events subject to a set of different constraints. These constraints may include both hard constraints that must be satisfied and soft

constraints which can be used to evaluate the solution quality. It is generally considered as a resource allocation problem in Operations Research, where resources of lecturers, students, classrooms and subjects are to be allocated into timeslots of a weekly timetable to achieve an objective function subject to constraints among resources (Schaerf, 1999a).

A large number of variants of the timetabling problem have been proposed in the literature, which differ from each other based on the type of institution involved and different constraints. One specialized area of timetabling is the school or university timetables. Schaerf (1999a) classifies the timetabling problem into three main classes based on the type of institution involved and the type of constraints:

- School timetabling. Scheduling for all the classes of a high school avoiding teachers and groups of student double booking weekly. The students are grouped in classes with similar study plans;
- University timetabling (course timetabling). Scheduling of all the lectures of a set of university degree modules, minimizing the overlaps of lectures having common students and avoiding lectures double booking;
- Examination timetabling. It is used to avoid student double booking examination.

Several other problems are also associated with the more general timetabling problem including room allocation, meeting scheduling, staff allocation such as bus or rail timetabling and employee timetabling.

Various methods such as graph colouring problem (GCP) (Almond, 1966; Vit'anyi, 1981; de Werra, 1985), integer linear programming techniques (Tripathy, 1984), simulated annealing (Abramson, 1991a), tabu search (Hertz, 1992; Costa, 1994) and genetic algorithms (Abramson and Abela, 1991b; Burke et al., 1994; Terashima-Marín, 1998) have been employed to solve a variety of educational timetabling problems. However the models formulated by some of these techniques cannot be easily reformulated or customized to support changes. The following sections discuss various methods of solving the timetabling problem.

2.3 Operations Research

In the early days of educational timetabling research, the most common approach to modelling and solving the timetabling problem has been to employ graph colouring problem (GCP) (Almond, 1966; Vit'anyi, 1981; de Werra, 1985). Other research which using integer linear programming techniques to represent the timetabling problem has been also carried out (Tripathy, 1984).

The GCP is a well-researched problem. Several efficient graph colouring heuristics have been devised. The GCP is to colour the vertices of a graph G= (V, E), where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices and E the set of edges connecting

the vertices to find a colouring *C: V -> **N*** such that connected vertices always have different colours. The optimal solution for the GCP is to find the minimum number K such that a feasible K-colouring exists.

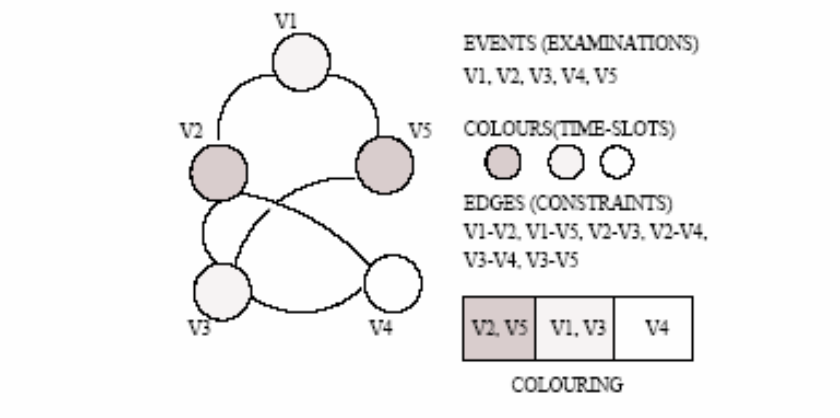


Figure 2.1 Equivalence between GCP and timetabling problem
(Source: Terashima-Marín, 1998, p.17)

Timetableling problems in their simplest form can be viewed as a GCP, where each node represents a task, each colour represents a timeslot, and each edge (v_i, v_j) indicates that v_i and v_j should not be placed within the same timeslot. Figure 2.1 shows the equivalence of the GCP and timetabling problem. Figure 2.1 uses the vertices V_1, V_2, V_3, V_4, and V_5 to represent the lectures or the examinations. The different colours mean different timeslots. The edges are the constraints. For example, V_1- V_2 means V_1 and V_2 cannot be held on the same time. But V_2 and V_5 have the same colour, so they can be held on the same time. Finally the objective of the problem is to find a solution to satisfy all the constraints: V_1 and V_2, V_1 and

V_5, V_2 and V_3, V_2 and V_4, V_3 and V_4, V_3 and V_5 cannot be held on the same time. However V_2 and V_5, V_1 and V_3 can be held at the same time.

Even though many related research is dependent on the GCP approach to get feasible solution (Almond, 1966; Vit'anyi, 1981; de Werra, 1985), de Werra (1997) concludes that the mathematical programming formulations may be useful in characterizing solvable cases of generally NP-complete extensions of colouring problems. The GCP approach can solve timetabling problem very precisely for mathematical programming characteristic. The GCP technique adapts well to small-scale problems, however they fail to scale up for larger ones (Tripathy, 1984). Normally the real timetabling problem is a big-scale problem, so the timetabling problem solved by the GCP approach is still far from the real situations encountered in timetabling.

2.4 Human-machine interaction

This involves an iterative approach (Mulvey, 1982; Chahal and de Werra, 1989; Franklin et al., 1995), with the machine proposing an initial solution and the user accepting it, improving upon it manually or instructing the machine to provide another solution. The process iterates until the user is satisfied with the solution or no further implement is possible. Mulvey (1982) proposes the following model for human-machine interaction: approximation, evaluation and modification. The user requires an approximate model of the ill-defined problem; subsequently, this

model has to be evaluated by an estimator as to whether it proves to be a good starting solution and how flexible it is in aiding the user to improve upon the solution. The user may attempt to get the solution by modifying it by hand. If he feels that the solution determined by the approximate model is not close to an acceptable schedule, he may redo these manual alterations until an acceptable solution is obtained or until he wishes to be aided by the approximate model. Finally, the user or the machine goes through the modification process as outlined above.

Mulvey (1982) presents a general solution strategy of the human-machine interaction model. Figure 2.2 shows the flowchart for the human-machine interaction strategy, which is described step-by step as follows:

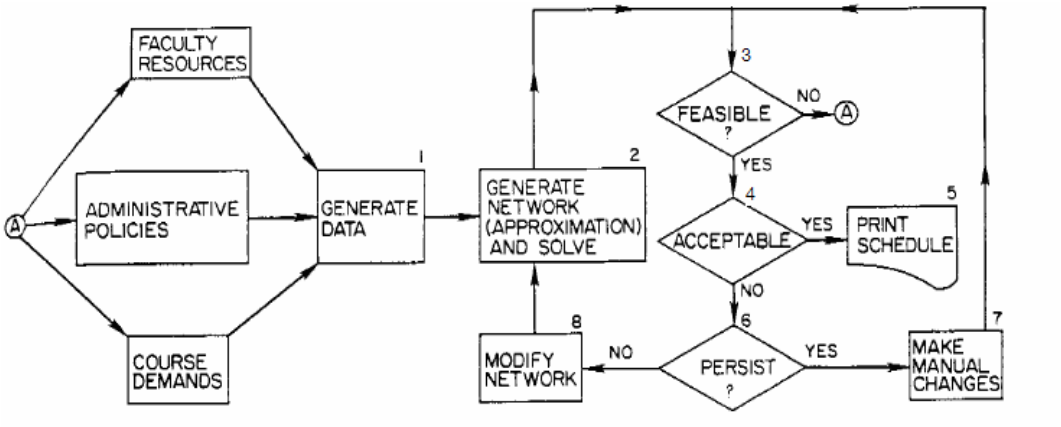


Figure 2.2 The human-computer interaction solution strategy (Source: Mulvey, 1982, p.69)

1. Generate the data required for the approximate model. First, the machine will ask user to input the data, such as administrative polices, faculty

resources and course demands. This data can be all modified to reflect the administrative policies. Then the machine will generate the data to comply user's requirement.

2. (Approximation) After that, the machine will generate network mode and solve.
3. (Evaluation) Determine whether the candidate scheduled is a feasible alternative. If not, return to step 1.
4. (Evaluation) Determine whether the candidate schedule is acceptable. If not, go to step 6.
5. Print the schedule.
6. Decide whether to persist in attempting to improve the candidate schedule by hand. If not, go to step 8.
7. (Modification) Make manual changes in the candidate schedule. Go to step 3.
8. (Modification) Make changes in the network formulation. Go to step 3.

The major drawback of such implemented systems is that for large problems, the process can be laborious, and obtaining a good approximate model may be computationally expensive (Gunadhi et al., 1996).

2.5 Artificial Intelligence

There are various meta-heuristic methods such as simulated annealing (Abramson,

1991a), tabu search (Hertz, 1992; Costa, 1994) and genetic algorithms (Abramson and Abela, 1991b; Burke et al., 1994; Terashima-Marín, 1998) that have been employed to solve a variety of educational timetabling problems.

2.5.1 Genetic Algorithms

Genetic algorithms form a class of algorithms used to find approximate solutions to difficult problems through application of the principles of evolutionary biology, such as inheritance, mutation and natural selection. Genetic algorithms mimic the process of natural selection and can be used as a technique for solving complex optimizations problems, which have very large search spaces. Unlike many heuristic schemes, which only have one sub-optimal solution at any time, genetic algorithm maintains many individual solutions in the form of a population. Individuals (parents) are chosen from the population and are then mated to form a new individual (a child). The child is further mutated to introduce diversity into the population.

The following description is taken from Burke et al. (1994). A genetic algorithm starts by generating a set (population) of timetables randomly. These are then evaluated according to some sort of criteria. On the basis of this evaluation population members (timetables) are chosen as parents for the next generation of timetables. By weighting the selection process in favour of the better timetables, the worse timetables are eliminated while at the same time the search is directed

towards the most promising areas of the search space. This process is shown as in Figure 2.3.

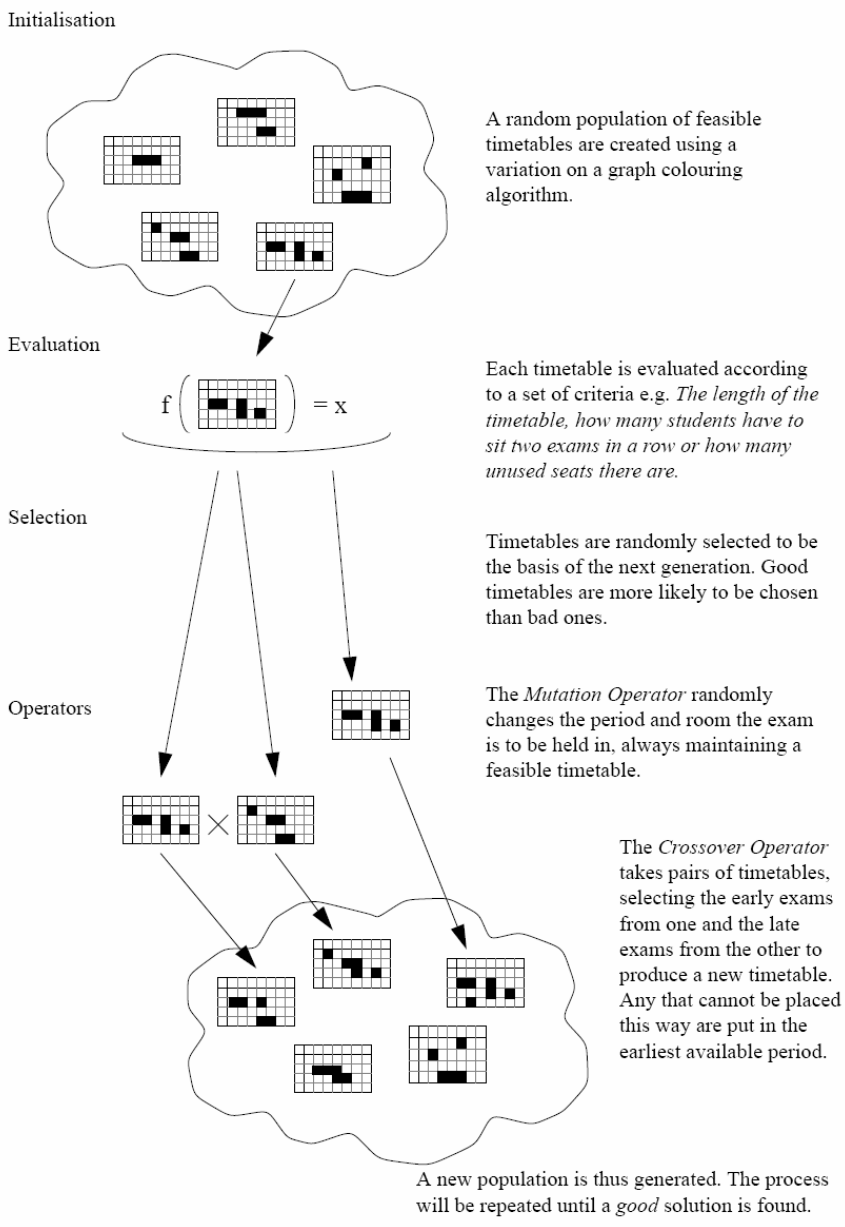


Figure 2.3 The genetic algorithm process (Source: Burke et al., 1994, p.56)

According to Abramson and Abela (1991b), genetic algorithms require a measure of the fitness of an individual in order to determine which individuals in the

population survive. The higher the fitness, the more likely the individual will survive. The cost measure developed for the timetabling problem represents a high quality solution with a minimal cost value.

2.5.2 Tabu Search

Tabu search is a meta-heuristic method to solve global optimization problems based on multi-level memory management and response exploration. It can be applied to solve the timetabling problem. The basic concept of Tabu Search as described by Glover (1986) is “a meta-heuristic superimposed on another heuristic”. Specific applications to timetabling problem have been presented by Hertz (1992) and Costa (1994).

Tabu search is a general search procedure devised for finding a global minimum of a function *f* defined on a feasible set *x*. For each solution *s* in *x*, we define a neighbourhood *N(s)* which consists of all feasible solutions that can be obtained by applying to *s* a simple type of modification *m*. The procedure starts from an initial feasible solution and tries to reach a global optimum solution by moving step by step. It is an iterative improvement algorithm that replaces the initial solution with the candidate solution at each stage of the algorithm even if the candidate has conflicts with the initial solution (Costa, 1994).

We will describe how tabu search is used to solve the timetabling problem based on the following steps (Hertz, 1992). As we discussed the timetabling problems

can be formulated as graph colouring problems. The lectures are the vertices of a graph, the periods correspond to the colours and two vertices are linked by an edge if the corresponding courses cannot be given at the same time. Tabu search has been successfully applied to this problem by extending its adaptation to the graph-colouring problem. A feasible solution is defined as a schedule that satisfies a subset C of constraints and a neighbour solution is obtained by modifying the schedule of one lecture. If all lectures last one period, this is equivalent to change the colour of a vertex. Let us consider two lectures A and B involving common students. If A cannot be scheduled at period p for some reason, then we shall never visit a solution where A is given at time p. However, if this constraint does not exist and B is scheduled at time p, then we are allowed to schedule another A at time p, even if this violates the constraint of no overlapping of lectures involving common students. The reason is that A is possibly given at time p in an optimal schedule and the violation of the constraint can be cancelled by moving course B to another period. Unfortunately, the tabu search is not effective enough in a big problem space (Hertz et al., 1997).

2.5.3 Simulated Annealing

Abramson (1991a) uses a simulated annealing technique to solve timetable problem. The simulated annealing technique simulates the cooling of a collection of hot vibrating atoms. When the atoms are at a high temperature they are free to

move around and tend to move with random displacements. However, as the mass cools the inter-particle bonds force the atoms together. When the mass is cool, no movement is possible and the configuration is frozen. If the mass is cooled quickly then the chance of obtaining a low cost solution is lower than if it is cooled slowly (or annealed). At any given temperature a new configuration of atoms is accepted if the system energy is lowered. However, if the energy is higher, then the configuration is accepted only if the probability of such an increase is lower than that expected at the given temperature. The application of simulated annealing to the timetabling problem is relatively straightforward. The atoms are replaced by elements. The system energy is replaced by the timetable cost. An initial allocation is made in which elements are placed in a randomly chosen period. The change in cost is calculated from two components: i) the cost of removing the element from the period; and ii) the cost of inserting the element to the period.

The cost of removing an element consists of a class cost, a teacher cost and a room cost. Likewise, the cost of inserting an element consists of a class cost, a teacher cost and a room cost. If after removing an element from a period the number of occurrences of that class is greater than 0, then the class cost saving is 1. Similarly, if there is one or more occurrences of the teacher after that teacher has been removed then the teacher saving is 1. This technique also applies to rooms. The cost of inserting an element can be calculated using the same basic

technique. In this way it is possible to determine the change in cost incrementally without recalculating the cost of the entire timetable. This attribute is particularly useful when the parallel version of the algorithm is implemented (Abramson, 1991a).

It may be necessary to express a preference that a requirement appears in a particular period or range of periods. This information can be easily added into the cost function, including a cost component, which reflects how close a requirement, is to its desired period. When the requirement is close to its preferred period the cost is low, however, as the requirement moves further from the period the cost is increased. A preference for more than one period may be indicated, so that a number of preferred times could be nominated. This cost component can be computed very quickly by looking up the preference in a statistically computed cost table. One of the advantages of simulated annealing over algorithms which always seek a better solution is that simulated annealing is less likely to get caught in local minima because the cost can increase as well as decrease (Abramson, 1991a).

As with genetic algorithms, a major advantage of simulate annealing is its flexibility and robustness as a global search method. It can deal with highly nonlinear problems and non- differentiable functions as well as functions with multiple local optima. It is also amenable to parallel implementation. A disadvantage is that simulate annealing methods are computation-intensive (Busetti, 2003).

2.6 Constraint Satisfaction Problem (CSP)

The roots of constraint programming can be traced back to the work on constraint satisfaction problems in the 1970s (Montanari, 1974; Mackworth, 1977; Freuder, 1978), but their widespread study and application started since the end of 80's when the extension of unification algorithm of logic programming (O'Keefe, 1990) led to the proposal of constraint logic programming scheme (Van Hentenryck, 1989). CSP has been applied to university and college timetabling (Brittan and Farley, 1971; Jaffar and Maher, 1994; Guéret et al., 1996; Lajos, 1996; Deris et al., 1997; Abdennadher and Marte, 2000).

The one unique feature of the CSP is it can divide the constraints to both hard and soft nature constraints. For a scheduling solution, hard constraints are conditions that must be satisfied, soft constraints, however, may be violated, but should be satisfied as much as possible. The first level of the constraint programming architecture allows users to state constraints over these programming variables. The second level of this architecture allows users to write a computer program that indicates how the variables should be modified so as to find values of the variables that satisfy the constraints.

The CSP has the NP-complete character. It has been applied in the area of planning, scheduling such as job-shop problem (Chen and Smith, 1997); car

sequencing problem (Van Hentenryck et al., 1992); vehicle routing problem (Gendrau et al., 1997) and rostering problem (Ryan, 1992).

Timetabling problems is a type of assignment problems with large amount of complex constraints, thus usually can be easily modelled as CSPs (Brailsford et al., 1999). The application for solving the timetabling problem using constraint satisfaction programming approach allows the formulation of all the constraints of the problem in a more declarative way than other approaches (Lajos, 1996; Guéret et al., 1996). Thus the CSP is particularly well suited for timetabling problems, since it allows the formulation of all constraints of the problem in a more declarative way than other approaches. Although its performance can be greatly affected by minor changes in problem formulation, the alternatives in formulation are still within the bounds of logical equivalence (Geske, 1998).

2.6.1 Definition

The CSP has three components: variables, values, and constraints. The goal is to find assignments of the variables to their candidate values such that all the constraints are satisfied (Nadel, 1989). There is no restriction on the type of each decision variable; hence decision variable can take on integer values, real values, set elements, or even subsets of sets.

Barták (1998) defines the CSP is a problem where one is given (X, D, C):

- a finite set of variables $X=\{x_1,...,x_n\}$,
- a function D_i which maps every variable x_i to a finite set of possible values (its domain),
- a finite set of constraints C_i restricting the values that the variables can simultaneously take.

The CSPs are an important class of combinatorial optimization problems. A solution to a CSP is simply a set of values of the variables such that the values are in the domains of the variables and all of the constraints are satisfied. An assignment of a unique domain value to each member of some subset of variables is called an instantiation (Dechter, 2003). There are two approaches to solving the CSP. One is using consistency technique, which is based on removing inconsistent values from variables' domains until the solution is found. The other is using the search algorithms.

2.6.2 Consistency

The problem of the existence of solutions in a constraint network is NP-complete. Hence consistency techniques have been widely studied to simplify constraint networks before or during the search of solutions. The CSP problem reduction has two functions: to reduce the problem to one, which is hopefully easier to solve, and to recognize insoluble problems. The whole idea of problem reduction is about removing redundant values. The question is how to identify such values.

Over the years, a number of consistency concepts have been developed to help in identifying redundant values. The consistency concepts are defined in such a way that if the presence of a value in a domain, then it can be deduced to be redundant (Dechter, 2003).

Arc-consistency is the most widely used method originating from Waltz (Waltz, 1975), who developed it for vision problems. Ar-consistency has also been studied by Mackworth and Freuder (Mackworth, 1977; Mackworth and Freuder, 1985), who have proposed an algorithm having an optimal worst case time complexity O (ed^2), where e is the number of constraints and d the size of the largest domain.

2.6.2.1 Arc-consistency

Arc (V_i, V_j) is said to be arc consistent if for every value x the current domain of V_i there is some value y in the domain of V_j such that V_i=x and V_j=y is permitted by the binary constraint between V_i and V_j. Note that the concept of arc-consistency is directional, i.e., if an arc (V_i, V_j) is consistent, then it does not automatically mean that (V_j, V_i) is also consistent. We will use the following example from Dechter (2003) to demonstrate the concept of arc-consistency.

Example 1:

Consider two variables x and y whose domains are $D_x = D_y = \{1, 2, 3\}$ and the constraint R_{xy} expressing the relation x < y. We can describe the constraint R_{xy}

where the domain of each variable is an enclosed set of points, and there are arcs connecting points that correspond to consistent pairs of values. In this example, when the value $3 \in D_x$ has no consistent matching value in D_y, then the constraint R_{xy} is not arc-consistent relative to x. Similarly R_{xy} is not arc-consistent relative to y, since y = 1 has no consistent match in x.

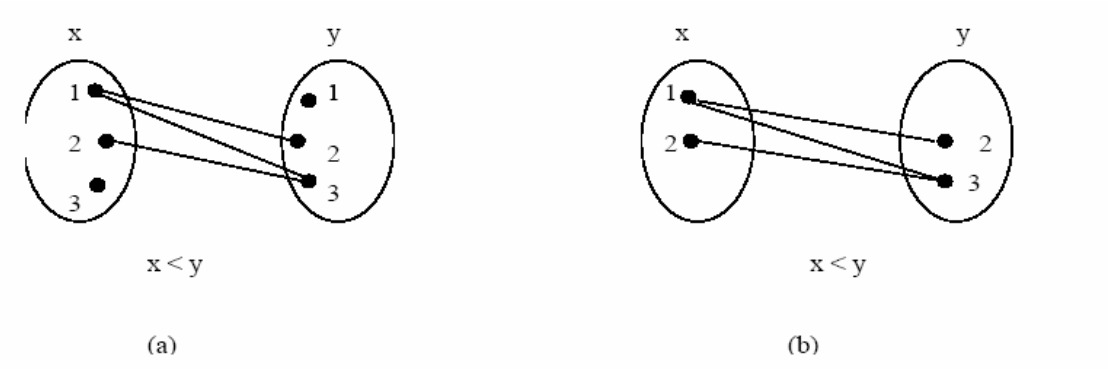


Figure 2.4 Example of arc consistency (Source: Dechter 2003, p.55)

If, however, we eliminate the domains of both x and y such that $D_x = \{1, 2\}$ and $D_y = \{2, 3\}$, then x is arc-consistent relative to y, and y is arc-consistent relative to x. The matching diagram of the arc-consistent constraint network is shown in Figure 2.4b.

A series of algorithms, AC-1, AC-2, AC-3 and AC-4 have been formulated to achieve arc-consistency (Dechter, 2003). AC-1 is simple but inefficient. Procedure REVISE is often used in the AC algorithm. The pseudo-code for REVISE is given as follows:

Procedure REVISE(x,y)

```
for each v_x in domain of x do
if there is no v_y in the domain of y such that
   (v_x, v_y) is consistent
then delete v_x from the domain of x
```

AC-1 repeatedly applies REVISE to all constraints in both directions until no changes are made (Mackworth, 1977). The pseudo-code for AC-1 is given as follows:

Procedure AC-1(G)

```
Let Q be the set of arcs of G (not self-cyclic)
repeat
for each (x,y) in Q do
REVISE(x,y)
REVISE(y,x)
until no domain is changed
```

The inefficient of AC-1 is that one revision causes all arcs to be revised on next cycle. The runtime of AC-1 is O (end^3) for graph with n variables with domain size bounded by d, and e binary constraints. Algorithm AC-1 can be improved; there is no need to process all the constraints if only a few domains were reduced in the previous round. The idea behind the next version of arc-consistency is to maintain a queue of constraints to be processed. This idea is improved and results in various AC algorithms such as AC-2, AC-3, AC-4 and so on.

2.6.2.2 Path Consistency (k-consistency)

Given a constraint network R = (X, D, C), a two variable set { x_i, x_j }is said to be path-consistent relative to variable x_k if and only if for every consistent assignment (< x_i, a_i >; < x_j, a_j >) there is a value $a_k \in D_k$ such that the assignment (< x_i, a_i >; < x_k, a_k >) is consistent and (< x_k, a_k >; < x_j, a_j >) is consistent (Dechter, 2003). We will again use an example from Dechter (2003) to illustrate this concept.

Example 2:

Consider the network of three variables {x, y, z} where each variable has the domain {1,2}, that has two equality constraints: R_{xz}: x = z, and R_{yz}: y = z.

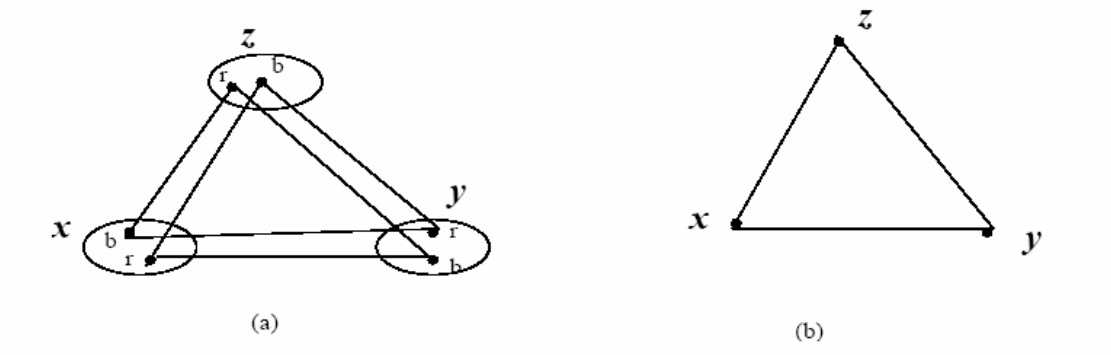


Figure 2.5 Example of path-consistency (Source: Dechter 2003, p.63)

To make this network path-consistent, the constraint $R_{xy:}$ x = y should be inferred and added to the network.

The concept of path consistency can be extended to k-consistency. Given a general network of constraints R = (X, D, C), a network is k-consistent if and only

if given any consistent instantiation of any (k-1) distinct variables, there exists an instantiation of any k^{th} variable such that the k values taken together satisfy all of the constraints among the k variables (Dechter, 2003).

2.6.3 Search algorithms

Most algorithms for solving the CSPs search systematically through the possible assignments of values to variables. Such algorithms are guaranteed to find a solution if one exists, or to prove that the problem has no solution but they may take a very long time to do so. Three systematic search algorithms will be presented here: a simple backtracking algorithm, look ahead strategy and look back strategy. The first, a simple backtracking algorithm, is generally not used in practice, because in most cases it is very inefficient, but it is presented here for comparison with the more sophisticated algorithms of the look ahead scheme and the look back scheme.

2.6.3.1 Backtracking

Backtracking is the most common algorithm for performing systematic search. In the backtracking algorithm, the current variable is assigned a value from its domain. This assignment is then checked against the current partial solution. If any of the constraints between this variable and the last variables is violated, the assignment is abandoned and another value for the current variable is chosen. If

all values for the current variable have been tried, the algorithm backtracks to the previous variable and assigns it a new value.

One of the disadvantages of this algorithm is thrashing. Thrashing refers to repeated failure due to the same reason. Thrashing can be avoided by intelligent backtracking. It works by a scheme on which backtracking is done directly to the variable that caused the failure. Another disadvantage of this method is redundant work. This means conflicting values of variables are not remembered. There is a backtracking based method that eliminates both of the above drawbacks of backtracking. This method is traditionally called dependency-directed backtracking and is used in the truth maintenance systems. Another disadvantage is it detects the conflict too late as it is not able to detect the conflict before the conflict really occurs. This drawback can be avoided by applying consistency techniques to forward check the possible conflicts (Kumar, 1992).

2.6.3.2 Look ahead scheme

The look ahead schemes are invoked whenever the algorithm is preparing to assign a value to the next variable (Dechter, 2003). There are two approaches in look ahead schemes: forward checking and look ahead maintaining arc consistency.

Forward checking is the easiest example of look ahead scheme (Haralick and Elliott, 1980). It detects the inconsistency earlier than simple backtracking. It allows branches of the search tree that will lead to failure to be pruned earlier than with simple backtracking. This reduces the search tree and the overall amount of work done. But it should be noted that forward checking does more work when each assignment is added to the current partial solution. The forward checking algorithm checks the constraints between the current and past variables and the future variables. When a value is assigned to the current variable, any value in the domain of a future variable which conflicts with this assignment is removed from the domain. The advantage of this is that if the domain of a future variable becomes empty, it is known immediately that the current partial solution is inconsistent, and as before, either another value for the current variables is tried or the algorithm backtracks to the previous variable. The state of the domains of future variable, as they were before the assignment which led to failure, is restored. With simple backtracking, this failure would not have been detected until the future variable was considered and it would then have been discovered that none of its values were consistent with the current partial solution (Barták, 1998).

We will use the 8 queen problem to demonstrate the forward search schema. The n-queens problem requires placing n queens on an NxN chessboard in such a way that no queen can take any other: this means no two queens can be on the same row, the same column or the same diagonal of the board. The notation of (row,

column) is used here to illustrate the row and column of the queen put on the chessboard. Figure 2.6 shows the chessboard after (4, B) is put on. All the squares, which have conflict with at least one of the committed labels, are marked by “x”. One can easily see from Figure 2.6 that no square in row 6 is safe, and therefore, it is not necessary to go on with the current four queens. On the other hand, when the simple backtracking is used, columns D and H of Queen 5 will be tried before the program concludes that no position is available to put a queen in row 6 (Tsang, 1993). This example shows the effect of forward checking scheme. It can deduce search space by detecting the conflict earlier. In this case, the label (4, B) should be rejected because all the values for Queen 6 are incompatible with the committed labels.

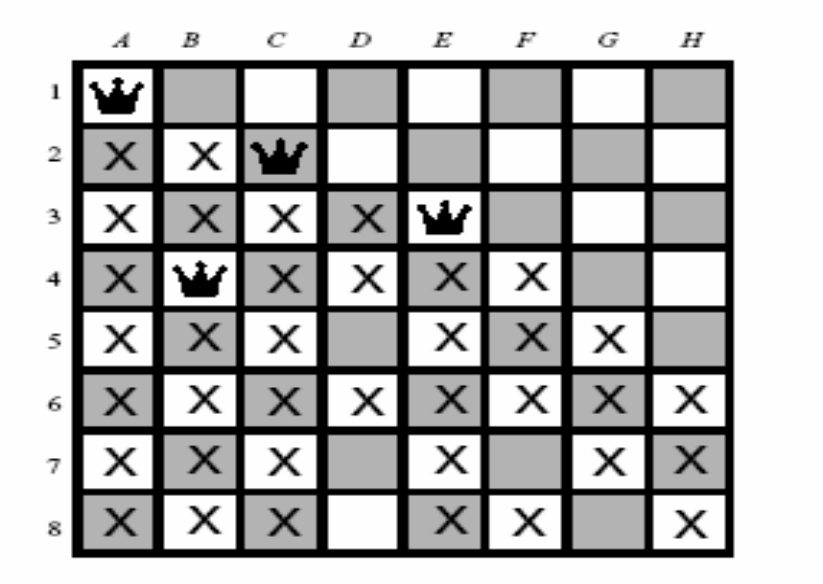


Figure 2.6 An example of forward checking (Source: Tsang 1993, p.125)

Now we will discuss the full look ahead maintaining arc consistency (MAC) schema (Sabin and Freuder, 1994). Forward checking checks only the constraints between the current variable and the future variables. The advantage of MAC is that it detects also the conflicts between future variables and therefore allows branches of the search tree that will lead to failure to be pruned earlier than with forward checking.

The example illustrates in Figure 2.7 shows the situation after the first three queens have been placed on an 8-queen problem. The following is one possible scenario after (1, A), (2, C) and (3, E) have been put on. As in MAC, B can be removed from the domain of Queen 4 and D can be removed from the domain of Queen 5 at this stage, as they are both incompatible with the only value D for Queen 6. These labels are marked “O”. Similarly, the labels (7, D), (8, B), (8, D) and (8, F) will be deleted at this stage as well, as they have no compatible values with Queen 6. After (4, B) is deleted, G and H are the only values left for Queen 4. MAC will delete (5, H), as it is incompatible with any of the remaining values for Queen 4. Then (7, B) will be removed for having no compatible value with Queen 5. Finally, (7, F) conflicts with (8, G), which are the only values left for Queens 7 and 8, respectively. After we delete the value (8, G), it leaves the domain of Queen 8 to be empty. Thus there is no solution for the 8 Queen problem when the queen have been put on (1, A), (2, C) and (3, E). The MAC

strategy will try to find another solution and the advantage of it is if can reduce the search space (Tsang, 1993).

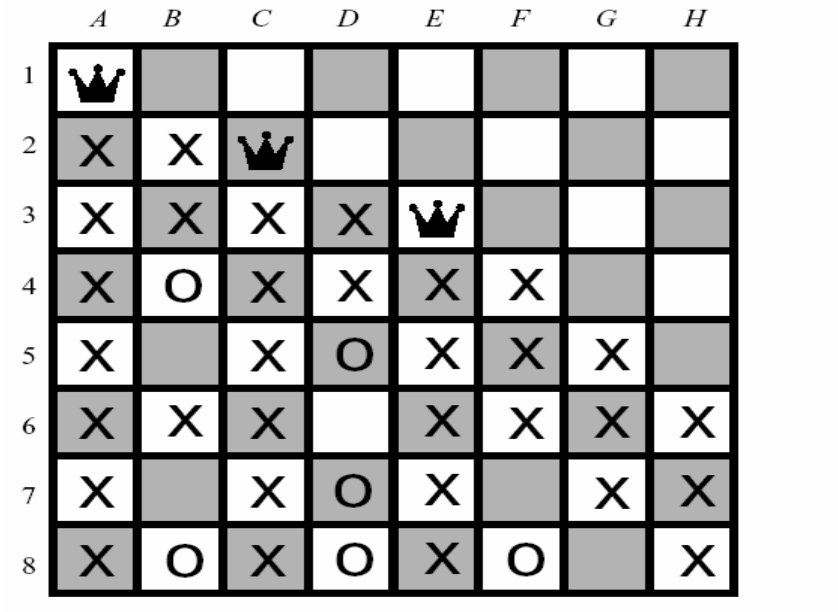


Figure 2.7 An example of MAC (Source: Tsang, 1993, p.134)

2.6.3.3 Look back scheme

The look back schemes are invoked when the algorithm encounters a dead-end and prepares for the backtracking step (Dechter, 2003). All look back schemas share the disadvantage of late detection of the conflict. They solve the inconsistency when it occurs but do not prevent the inconsistency to occur. All the look back algorithms analyse the reasons of failures so as to jump back to the culprit decisions remember and deduce redundant compound values. Hence failure experience can help in future search (Tsang, 1993). We will discuss the approach of back-jumping and back-marking in the look back schema.

Like backtracking, back-jumping picks one variable at a time. Given a variable, back-jumping finds a value for it, making sure that the new assignment is compatible with the labels recorded so far. It backtracks if no value can be assigned to the current variable. However, when back-jumping needs to backtrack, it analyses the situation in order to identify the "culprit decisions" which have caused the failure. If every value in the domain of the current variable is in conflict with some committed labels, then back-jumping backtracks to the most recent culprit decision rather than the immediate past variable, as is the case in backtracking (Tsang, 1993).

We will again use the 8-queens problem to illustrate the back-jumping. Figure 2.8 shows a situation after five queens have been placed onto the board. When Queen 6 is looked at, it is found that there is no solution. The back-jumping realizes that changing the value of Queen 5 will not resolve the conflicts. The back-jumping will try to find the earliest queen which lead to the incompatible result. For each square in the domain of Queen 6, the earliest queen who is incompatible with it is identified. We call these identified queens "culprit queens". For example, the culprit queen for (6, B) is Queen 3 and Queen 4. Back-jumping will then backtrack to the most recent of all the culprit queens. In this example, the queen that will be backtracked to is Queen 4. Unlike backtracking, back-jumping does not simply go back. It will not look at (5, H). If all the values for Queen 4 have

been exhausted after this backtracking, then the back-jumping will backtrack to Queen 3 (Tsang, 1993).

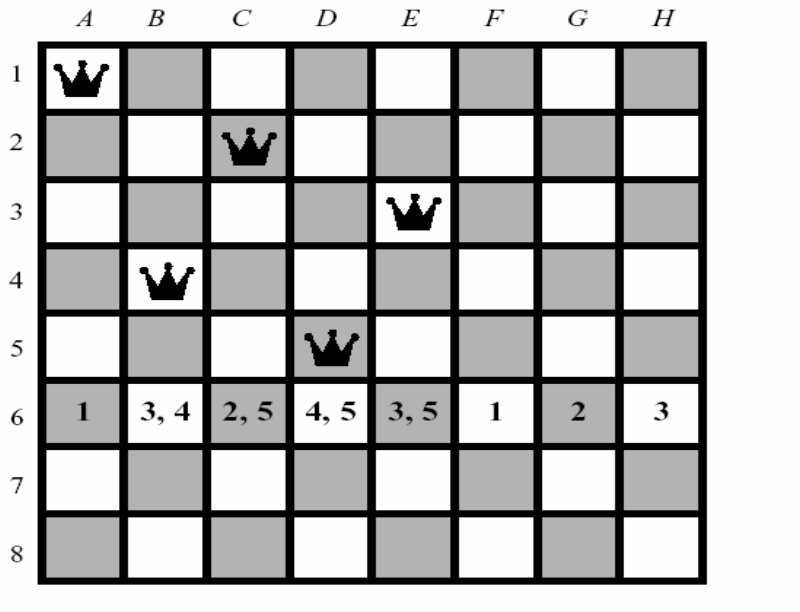


Figure 2.8 An example of back-jumping (Source: Tsang 1993, p.138)

Back-marking reduces the number of compatibility checks by remembering for every label the incompatible labels, which have already been committed to. It avoids repeating compatibility checks which have already been performed and which have succeeded (Tsang, 1993). For each variable, back-marking records the highest level that it last backtracked to. This helps back-marking to avoid repeating those compatibility checks, which are known to succeed or fail.

Figure 2.9 shows a state in running back-marking on the 8-queens problem, assuming that the variables are labelled from Queen 1 to Queen 8. A chessboard situation for the 8-queens problem shows the values of Marks and LowUnits at some state during back-marking. The number in each square shows the value of

Mark for that square. At the state shown in Figure 2.9, five queens have been labelled, and it has been found that no value for Queen 6 is compatible with all the marked labels. Therefore, backtracking is needed. As a result, all the values of Queen 6 have been rejected, and consequently, LowUnit(6) has been changed to 5. If and when all the values of Queen 5 are rejected, both LowUnit(5) andLowUnit(6) will be changed to 4 (Tsang, 1993).

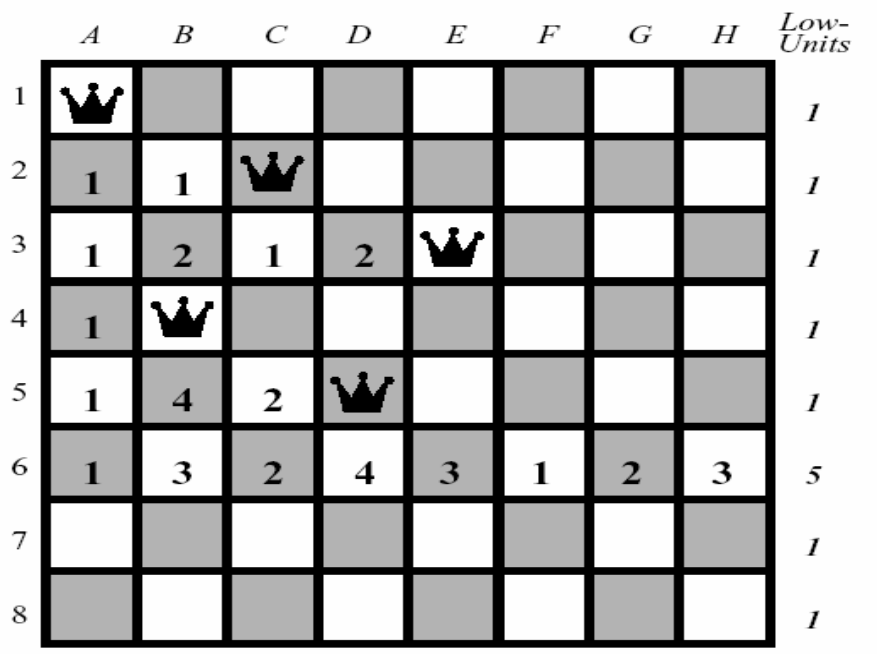


Figure 2.9 An example of back-marking (Source: Tsang 1993, p.151)

2.6.4 Example to compare backtracking and forward checking

The following 4-queen problem example will also give a function comparison between the backtracking and forward checking. An example of a search tree built by the backtracking algorithms is shown in Figure 2.10. As a CSP, this problem has four variables, representing the rows of the chessboard, and each variable has

domain {1…4} representing the columns of the chessboard. The representation is: a Q on a particular square been assigned the value corresponding to that column. Crosses, and a tick mark dead-end marks the solution is eventually found.

The backtracking algorithm only checks the constraints between the current variable and the past variables. On the contrary, the forward checking not only checks the constraints between the current variable and the past variables, but also checks the constraints between current variable and the future variable. When a value is assigned to the current variable, any value in the domain of a future variable, which conflicts with this assignment, is removed from the domain. Forward checking allows branches of the search tree which lead to failure to be pruned earlier than with simple backtracking.

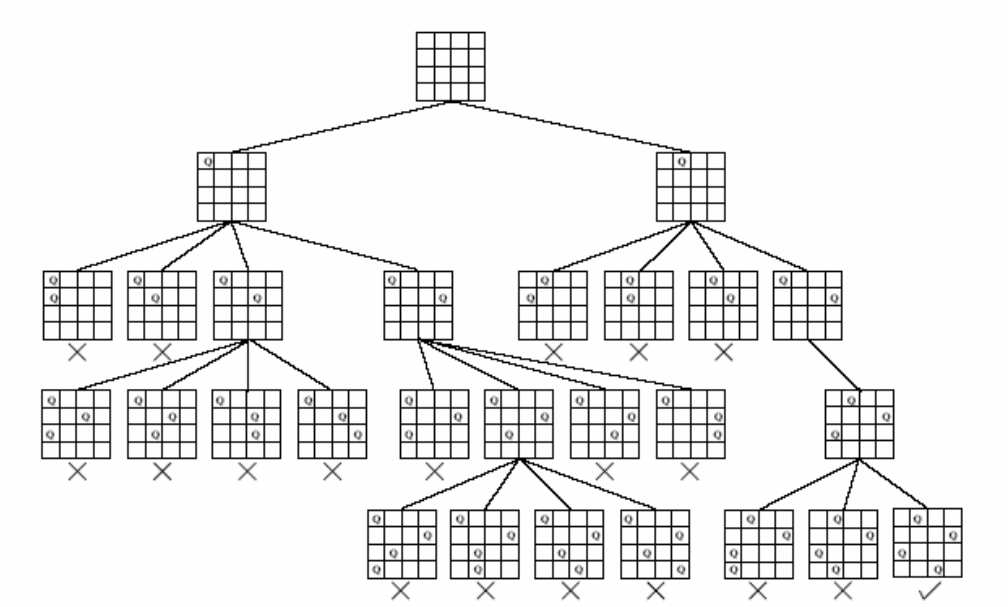


Figure 2.10 Search tree for 4-queens by backtracking (19 attempts) (Source: Barták, 1998)

If we start by placing a queen on the first row, then none of the other queens can be placed in the same column or in the same diagonal, and the values corresponding to the squares attacked by this queen can be removed from the domains of the variables representing the queens in rows 2 to 4. If the branch leads to a dead end, the first row queen has to be moved. The full search tree built by forward checking for 4-queen problem is shown in Figure 2.11. Squares with crosses denote values removed from the domains of future variables based on the past and current assignments.

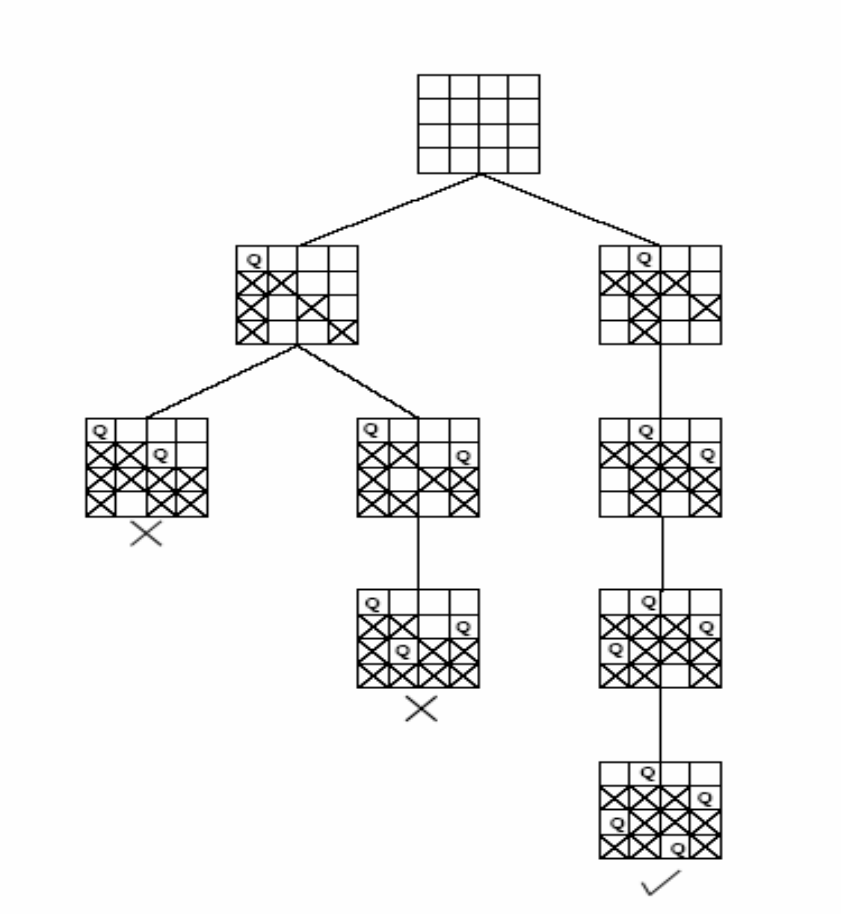


Figure 2.11 Search tree for 4-queens by forward checking (3 attempts) (Source: Barták, 1998)

In order to reduce the size of the search tree and to reduce the overall amount of work done, forward checking does more work when each assignment is added to the current partial solution.

2.6.5 Variable and Value Ordering

According to Kumar (1992), the performance of a backtracking algorithm can be improved in a number of ways, such as by performing constraint propagation at search nodes of the tree, by performing reason maintenance, intelligent backtracking or by choosing a good variable ordering or by choosing a good order for instantiation of different values of a given variable. In this section we will show how a good variable ordering can improve the performance of the search algorithms.

The variable ordering may be either a static ordering or dynamic ordering. In static ordering the order of the variables is specialized before the search begins and is not changed thereafter. In dynamic ordering, the choice of next variable to be considered at any point depends on the current state of the search (Haralick and Elliott, 1980). Dynamic ordering is not feasible for all tree search algorithms. For example, with simple backtracking there is no extra information available during the search that could be used to make a different choice of ordering from the initial ordering.

The order in which variables are considered for instantiation has a dramatic effect on the time taken to solve a CSP, as does the order in which each variable's values are considered. There are general principles, which are commonly used in selecting the variable and value ordering. For variable ordering the principle is “first-fail” and principle of value ordering is “first-succeed”. Decisions in the orderings could affect the efficiency of the search strategies significantly. The ordering in which the variables are labelled and the values chosen could affect the number of backtracks required in a search, which is one of the most important factors affecting the efficiency of an algorithm. In look ahead algorithms, the ordering in which the variables are labelled could also affect the amount of search space pruned. Besides, when the compatibility checks are computationally expensive, the efficiency of an algorithm could be significantly affected by the ordering of the compatibility checks (Tsang, 1993).

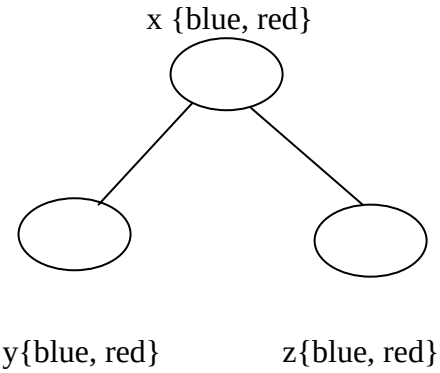
2.6.5.1 Variable Ordering

The most common variable ordering is based on the “first-fail” principle: “To succeed, try first where you are most likely to fail” (Barták, 1998). If failure is inevitable, the earlier we discover the better it is. On the other hand, if the current partial solution can be expanded to a complete solution, then every remaining variable must be instantiated and the one with smallest domain is likely to be the most difficult to find a value for. Therefore we need to choose the variable with the smallest domain, which can get the dead end earlier. The reason is that if the

current partial solution does not lead to a complete solution, then the current branch will eventually prove to be a dead-end.

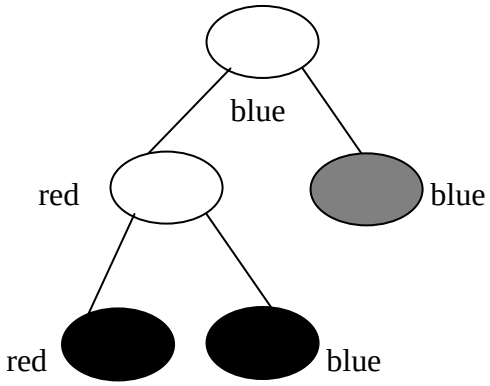
Consider a constraint network R having three variables x, y, z with domains D_x= {blue, red}, D_y= {blue, red}, D_z = {blue, red}. There is a constraint between x and the other two variables, which requires that the value assigned to x has the different colours with the values for y, z. The constraint graph of this problem is shown in Figure 2.12a. The graphs for the problem along the ordering d_1={x, y, z} and d_2={y, z, x} are give in Figure 2.12b and 2.12c. The nodes denote legal states. The black coloured ovals denote goal states and grey coloured ovals denote dead-ends. In this example, we use the principle of ordering to get the solution. If we want to choose a variable that fail first, then variable x should be chosen first because variable x has two constraints. The other two variables (y and z) only have one constraint. Thus ordering 1 will choose x first.

As illustrated in Figure 2.12, a constraint network can have different search spaces depending on the variable ordering. We can see that the search graph for ordering 1 includes 5 legal states, where the search graph for ordering 2 includes 7 legal states. Since the search space includes all solutions, one way to access whether its size is excessive is by counting the number of dead-end leaves. Ordering 1, for instance, renders a search graph with only one dead-end leaf, whereas the search graph for ordering 2 has 3 dead-end leaves. According to the “first-fail” principle, choosing ordering 1 (x, y, z) has less dead-end than ordering 2 (y, z, x).



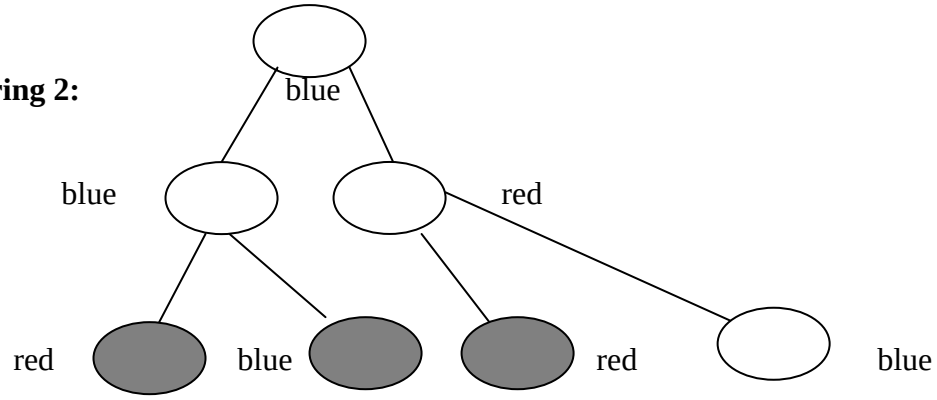
(a)

Ordering 1:



(b)

Ordering 2:



(c)

Figure 2.12 a) A constraint graph, b) search space along ordering 1, c) search space along ordering 2. (Colours next to the nodes represent value assignments)

2.6.5.2 Value Ordering

On the other hand, if we can find a complete solution based on past instantiations, we want to choose a value, which will lead to a solution. The principle of "Succeed First" is to choose a value which is likely a success, and unlikely to lead to a dead-end (Barták, 1998). Suppose we have selected a variable to instantiate, we should choose the value that can get be succeeded first. If we choose that values will never be succeeded thus this ordering will turn out to be a dead end. Therefore, we shall have to backtrack to the previous variable. In this case, every value for the current variable will eventually have to be considered, and the search space is the same.

We will use the same example as above to explain the value ordering principle. In current case, the domain of three variables x, y, z is D_x= {blue, red}, D_y= {red}, D_z= {red}. The graphs for the problem using the ordering 1={x, y, z} is shown in Figure 2.13.

We considered two cases. The first case is to select the blue colour first. The second case is to select the red colour first (see Figure 2.13a and 2.13b). The nodes denoted legal states. The black coloured ovals denote goal states, and grey coloured ovals denote dead-ends.

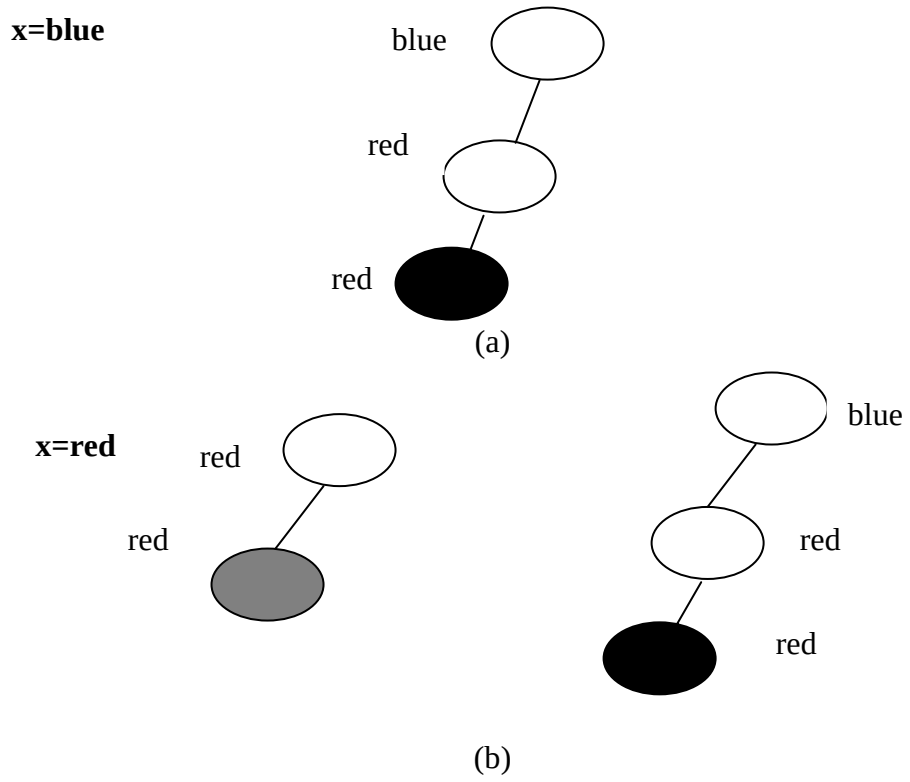


Figure 2.13 a) search space along x colour=blue, b) search space along x colour=red. (Colours next to the nodes represent value assignments)

In this example, we use the “succeed first” principle of value ordering to get the solution. If we want to choose a value that can succeed first, the value for variable x should choose blue first because the value has no conflict with the other two variables and can get a solution.

As illustrated in Figure 2.13, a constraint network can have different search spaces depending on the value ordering. We can see that the search graph for value= “blue” has three legal states, where the search graph for value= “red” has five legal states. When x= “blue”, the search graph with no dead-end leaf, whereas the search graph for x= “red” has one dead-end leaf. Thus by choosing x= “blue” first if has less dead end.

2.7 Conclusion

Timetabling as an example of a scheduling problem has become an application area with rich knowledge and experience. This chapter gives a literature review of basic solving techniques for timetabling problem.

CSP offers a new approach to solve the timetabling problem. The timetabling problem is NP complete and is a large-scale combinatorial problem with an extremely large search space. Thus, by solving the problem using the CSP approach, the search space can effectively be reduced to smaller search trees that lead to results within a finite amount of time. Constraints help to solve the problem in a finite amount of time, by stating various relationships between the variables. In this chapter we have examined the definition of the CSP and the approaches used to solve the CSP. We have also discussed the techniques of consistency and search algorithms to solve the CSP.

Chapter 3 Case Study

3.1 Introduction

The university course timetabling problem consists of scheduling a set of lectures and tutorials in a given number of rooms and time periods. The distinctive features of this class of timetabling problems are that lectures have common students and that availability and size of rooms plays an important role (Schaerf, 1999a). Most university timetabling problems are based on the basic student–course model (de Werra, 1985). Student attends a set of lectures. His/her selection is usually predefined by subscription of compulsory and optional subjects in universities. Lecture is taught one or more times a week during part of a year. Sometimes lectures are split to multiple lectures due to the large number of students subscribed to a subject or due to the design of the subject. Classroom of suitable size, equipment (laboratory, computer room, classroom with data projector, etc.) and location (part of building, building, campus, etc.) have to be assigned to each lecture. Thus, the problem consists of a set of subjects to be scheduled in timeslots, a set of rooms in which time can take place, a set of students who attend the subjects, and a set of subjects satisfied by rooms and required by timeslots.

Let

$S = \{s_1, s_2, \ldots s_n\}$ be the set of students;

$L = \{l_1, l_2, \ldots l_o\}$ be the set of subjects taught;

$T = \{t_1, t_2, \ldots, t_m\}$ be the available teaching periods ;

R = $\{r_1, r_2, \ldots, r_p\}$ be the set of rooms available.

where

S_l represents the set of students who take the subject l;

T_l is the set of timeslots allocated to the subject l;

R_l is the set of rooms assigned to the subject l.

Then $T_{l\,i}$ is the number of teaching periods for subjects l_i. S_{li} is the set of student's wishes to attend the subject l_i and R_{li} is the set of rooms that can be assigned to the subject l_i. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following constraints are satisfied:

- no student attends more than one subject at the same time;

  $\forall(S_{l\,i} = S_{l\,j}) \exists\ T_{l\,i} \neq T_{l\,j}$

  where $S_{li} = S_{lj}$ represents the student who take two subjects l_i and l_j, these two subjects can not be held at the same time.

- only one subject is in each room at any timeslot;

  $\forall(l_i = l_j) \exists\ T_{l\,i} \neq T_{l\,j}$

  where $T_{l\,i} \neq T_{l\,j}$ represents the timeslot is allocated to the subject l_i and l_j. When the two subjects l_i and l_j are held in the same room, they need to be held at different timeslots.

- the room size RSize(R_j) of room R_j is satisfied for all the features required by the subjects R_{li} and is big enough for all the attending students size SSize(S_{li});

  $\forall(R_{li} = R_j) \exists RSize(R_j) > SSize(S_{li})$

  where $SSize(S_{li})$ represents the size of students who attend the subject l_i.

- some of the timeslots have been reserved for special event E. Therefore, these timeslots should not be assigned any subjects;

  $\forall(T_j = E) \exists T_l \neq T_j$

  where $T_l \neq T_j$ means the set of timeslots is allocated to the subject l can not be equal to the special timeslot T_j when T_j is equal to special event E.

Thus the objective of the timetabling problem is to find a feasible solution that satisfies all the constraints. The timetabling problem in this research will be solved using the constraint programming approach. A major advantage of this approach is that it enables us to use the statement of a problem directly to develop a model for the problem. When we design a constraint-based model for a problem, we simply articulate the constraints themselves and then choose the variables with values that represent the solution of the problem. A timetable is essentially a schedule, which must suit a number of constraints. Constraints, in turn, are almost

universally broken into two categories: soft and hard constraints (Abdennadher and Marte, 2000). Hard constraints are constraints, of which, in any working timetable, there will be no breaches. For example, a student cannot be in two places at the same time. Soft constraints are constraints which may be broken, but of which breaches must be minimized. For example, there may be preferred hours in the form of the soft constraints in which a lecturer's classes might be scheduled, such as the tutorial preferably be scheduled on the same day as the lecture; the subjects be held on some preferred timeslot due to the preference or working contract nature of the lecturer.

The model we propose for a timetabling problem as a CSP is as follows: a timetable is a constrained variable the value of which is a function associating a value to each slot in time t. The timetable item is given by the set of subjects. Note that the subject can be offered as a lecture or a tutorial, which is considered as a timetable item. Basically our task consists in instantiation of the set of three tuples CSP (timetable item, classroom, time), i.e., each lecture or tutorial of a subject has assigned its set of classroom and time.

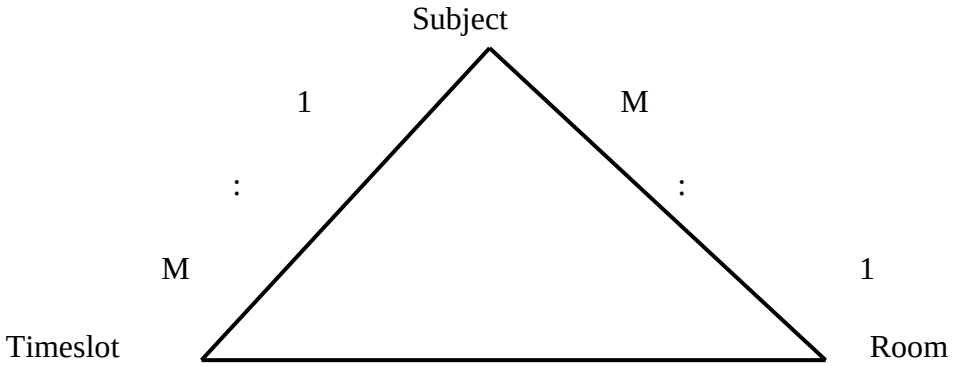


Figure 3.1: Assignment of rooms and timeslots to subjects

The variable is the room R and timeslot T for each subject S. The solution to this problem is the assignment of rooms and timeslots to a set of subjects on offer subject to satisfying the constraints. Figure 3.1 shows the relationship between subject and timeslot and that between room and subject. The relationship between subject and timeslot is one to many and that between room and subject is also one to many.

The rest of the chapter is organized as follows. Section 2 discusses ILOG Scheduler and ILOG Solver which are used to solve the timetabling problem in this research. Section 3 gives problem description of the sample case study used in this research and modelling of the problem as a CSP using ILOG.

3.2 Implementation using ILOG

We will use ILOG to solve the timetabling problem in our research. ILOG was created in 1987 to industrialise the expertise of INRIA (the National Institute for Research in Computer Science and Control), Europe's largest computer research centre in the field of symbolic computer languages and object-oriented environments (ILOG, 2004). It is a software development tool for object-oriented applications and is designed to be highly portable and be able to maintain code at a faster pace. ILOG decomposes the problem by separating the models from the search algorithms. This way, it is easy to change different algorithms applied to the same model. In addition ILOG can use multiphase search which allows a problem to split and solve in smaller parts (ILOG, 2004). We use two modules of ILOG in our implementation: Scheduler and Solver.

3.2.1 Scheduler

ILOG Scheduler is a modelling-based language. The model is built by an instance of the class `IloModel`, and each Scheduler class instance pertaining to that model (for example: activity, resource, and so forth) belong to an environment which is an instance of the class `IloEnv`. After the model is built, Solver is used to solve the problem. Thus it is important to build the model correctly and building a successful model depends on correctly implementing the Scheduler

object classes. The semantics of the ILOG Scheduler model are very simple. Basically the elements of the model consist of a schedule which includes both resources with limited capacity and activities. The Scheduler library proposes a simple object model for the representation of scheduling and resource allocation problems in terms of "resources" and "activities". An activity is a task to be completed in a schedule. Activities have a processing time. They execute over a specific interval of time in a schedule, which may be subject to temporal constraints. In general, activities require resources. Activities are represented by instances or combination of instances. An activity can be represented by `IloActivity`. A resource is an object such as a room, worker, machine, vehicle, supply, raw material, which adds value to a product or service in its creation, production, or delivery (ILOG, 2001c).

A scheduling problem can be viewed as a constraint satisfaction problem. A scheduling problem which can be defined by:

- A set of activities - tasks or work to be completed;
- A set of temporal constraints - definitions of possible relationships between the start and end times of the activities;
- A set of resources - objects such as workers, machines, vehicles, supplies, raw materials, rooms, etc. to allocate to the activities;
- A set of resource constraints - definitions of demands of the activities upon the resources-to link the activities to the resources.

ILOG Scheduler offers an object-oriented scheduling model that predefines the two main categories of constraints: temporal constraints and resource constraints. A set of temporal constraints indicate possible relationships between the start and end times of the activities. A temporal constraint is a condition imposed on the time interval during which an activity is processed. ILOG uses the temporal constraints like `startsAfterEnd` to represent the time constraints between two activities. A temporal constraint is logically expressed by a formula ($T_1 + t <= T_2$). T_1 and T_2 are variables representing start times or end times of activities, and at minimal delay elapse between the two time slots.

In Scheduler there are two types of temporal constraints: precedence constraints and time-bound constraints. Precedence constraints are instances of the class `IloPrecedenceConstraint`, and are used to specify when one activity must start or end with respect to the start or end time of another activity. Time-bound constraints, instances of the class `IloTimeBoundConstraint,` are used to specify when one activity must start or end with respect to a given time.

A resource constraint `IloResource` refers to a given resource is required in a given quantity for the execution of a given activity. A resource constraint specifies that the execution of a given activity requires a given resource. Scheduler provides several classes to represent resources. A root class for all the

resources is `IloResource`. `IloResource` has two subclasses, `IloStateResource` and `IloCapResource`. The class `IloCapResource`, which is the root class for all resources having some sort of capacity, has five subclasses. They are: `IloDiscreteResource`, `IloUnaryResource`, `IloDiscreteEnergy`, `IloReservoir` and `IloContinuousReservoir`. An instance of the class `IloStateResource` represents a resource of infinite capacity whose state can vary over time. Each activity may require a state resource to be in a given state or in any of a given set of states through its execution. Consequently, two activities that require inconsistent states cannot overlap. Figure 3.2 shows all the classes of `IloResource` (ILOG, 2001c).

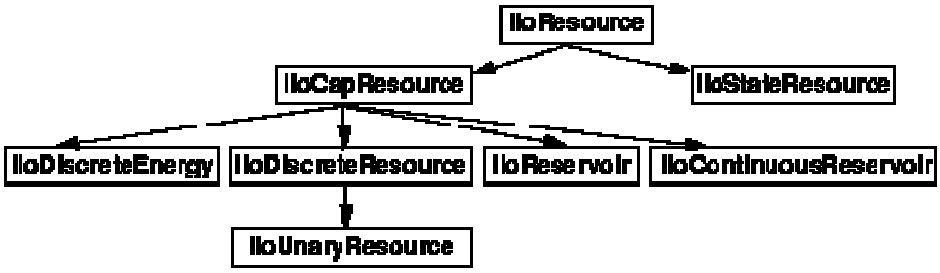


Figure 3.2 Classes of IloResource (ILOG, 2001c)

When we think of scheduling as a constraint satisfaction problem, our aim is to find a schedule that satisfies the constraints. Constraint propagation in ILOG occurs in both directions: from resources to activities and from activities to resources. The first case of resources to activities is to update the earliest and latest start and end times of activities. The second case of activities to resource is

to update the minimal and maximal capacities that can be used at any point in time.

As discussed above, Scheduler is used to build a model based on the problem definition. Modelling objects are extracted from the problem definition to the Solver and the Solver uses the extracted objects to solve all or part of the problem. At extraction time, the Scheduler Engine instances are created and initialized given the data in the parameters of the Scheduler instances. When used with Solver, the extraction of the modelling classes first creates corresponding instances of classes. The Scheduler extractor is the object that interprets the problem definition objects and creates the Schedule Engine objects for solving the problem. As with the Scheduler Engine, the Scheduler extractor is specific to Solver solution techniques. After the model is complete, it is extracted to the ILOG Solver.

3.2.2 Solver

ILOG Solver is the constraint-based optimization engine. It is a C++ object-oriented constraint-reasoning tool (ILOG, 2001a). Solver produces library for constraint definition and management. To use ILOG Solver, users need to define the variables and constraints then Solver automatically uses the posted constraints during the search for the solution. It reduces the domains of the constrained variables by removing those values that are inconsistent with the

constraints**.** In ILOG we use a prefix `Ilo` and `Ilc`. For each `Ilo` object (model object) in the model, there is a corresponding `Ilc` object (solver object). During the extraction of a model, the `Ilo` object will be extracted by the Solver to get an `Ilc` object.

The data types in ILOG are similar to any C++ programming. The type `IlcInt` corresponds to the C++ long type. The class `IlcIntVar` represents integer constraint variables. For example, a command like `IlcIntVar x (m, 0, 10)` creates a constrained variable x with the integers from 0 to 10 included in its domain, m is an object of the class `IlcManager` to which all constrained variables and constraints between them are connected. The method `IlcIntVar::removeValue(IlcInt a)` removes value `a` from a variable's domain and the method `IlcIntVar::setValue(IlcInt a)` assigns value `a` to a variable. The class `IlcIntVarArray` implements an array of constrained integer variables. For example, the call `IlcIntVarArray t (m, 10)` creates an array of 10 constrained integer variables which can be accessed with the overloaded operator [].

Constraints are represented by the class `IlcConstraint` and can be created with the overloaded C++ relational operators. For example, if x and y are objects of the class `IlcIntVar`, the call `IlcConstraint c(x>y)` creates a constraint c that ensures all values in `x's` domain are greater than the minimum of `y's` domain. In order for a constraint to be posted, the method

`IlcManager::add()` has to be used. The above constraint, for example, can be posted with the call `m.add(c)`. Posting a constraint ensures that it will be considered by the Solver's internal constraint propagation engine and that after any modification of a constrained variable, the constraint network will be modified, in order to be brought to a certain consistency degree. Solver uses a version of the AC-5 algorithm (Van Hentenryck et al., 1992) to bring the constraint graph to an arc-consistent state after any such modification. AC-5 is a generic arc-consistency algorithm and can be specialized for special constraints. AC-5 has been specialized to produce an O(ed) algorithm for a number of important classes of constraints where e is binary constraints with domain size bounded by d. The pseudo-code for AC-5 algorithm is given by Van Hentenryck et al. (1992) as follows:

Algorithm AC-5

```
Post : let P0=D10 x … x Dn0,
       P=D1 x … x Dn
G is maximally arc-consistent wrt P in P0.
Begin AC-5
       INITQueue(Q)
       For each(i,j)∈arc(G) do
       Begin
           ArcCons(i,j,△);
           Enqueue(i, △,Q);
           Remove (△,Di)
       End;
```

```
While not EmptyQueue(Q) do
Begin
    Dequeue(Q,i,j,w);
    LocalArcCons(i,j,w, △);
    Enqueue(i, △,Q);
    Remove(△, Di)
End
```

End AC-5

In the first step, all arcs are explored once and arc consistency is enforced on each of them. Procedure Remove (△, D) removes the set of values △ from D. The second step applies LocalArcCons on each of the element of the queue, possibly generating new elements in the queue.

In addition, ILOG Solver also provides a set of control primitives that allow user to implement his/her own heuristic search algorithm.

3.3 Description of sample case study

In the sample case study in this research, we have 24 subjects that made up of 203 timetable items to be scheduled into 12 rooms in 54 timeslots. A timetable item refers to either a lecture or a tutorial class. Background and problem description of the case study is described as follows.

A subject on offer is always made up of weekly lecture and tutorial. One or more lecturers can teach a subject. Each lecture usually runs for two hours. However a

lecturer may request to have a one-hour lecture only. There are two types of tutorial: classroom-based tutorial or laboratory-based tutorial. Both types of tutorial can either be one or two hour duration. In each case, the lecturer will specify the maximum number of students allowed to enrol in each tutorial group (this way the number of classes or tutorial groups that are required for the subject can be computed by the system). Other constraints under consideration are in the form of regulations such as lecture time can only be scheduled from 8 o'clock in the morning to 6 o'clock in the evening and tutorials not to be scheduled after 8pm. In addition, no lecture or tutorial must be scheduled between 1pm and 2pm on Wednesday to allow the teaching staff to attend meetings or seminars. In addition, a lecturer may make special requests so that individual requirements can be taken into consideration during the timetabling planning process. Example of such request includes a certain lecturer can only teach in a particular day or time of the week due to the nature of the employment such as part-time lecturer. In addition due to the way the subject is designed, a lecturer may make request such that a student can only take tutorial class after the lecture is conducted. Other examples of pre-specified requirements include a repeat lecture which caters mainly for part-time students must be held in the evening, and if a subject can only be taught by one lecturer then different tutorial groups cannot be scheduled concurrently. Table 3.1 show the sample data for the case study.

Subject code	Number of students enrolled	Lecture duration (hour)	Is repeat lecture required for part-time students?	Tutorial Type	Tutorial duration (hour)	Maximum number of students in a tutorial group	Cannot schedule concurrently with these subjects	Individual preferences
102	100	2	No	Laboratory	2	20		Not to have lecture before 11am, tutorial must be after the lecture
110	600	2	Yes	Classroom	2	40		One lecture must be scheduled in the morning and the repeat lecture in the afternoon
201	150	2	No	Laboratory	2	20	214, 311, 312	Tutorial has to be run after the lecture
214	240	2	Yes	Laboratory	2	20	201, 102	Two lectures to be held on different days
311	300	2	Yes	Laboratory	2	20	312, 315	Lectures to be scheduled either on Tuesday or Thursday. Tutorial can be on different days
312	320	2	Yes	Laboratory	2	20	311	Lectures to be scheduled either on Monday, Wednesday or Friday
315	100	2	No	Laboratory	2	20	311, 201, 312, 307	Tutorial must be held after the lecture
307	100	1	No	Classroom	2	20	311, 312	Lectures to be

								scheduled only on Wednesday
317	250	2	Yes	Laboratory	2	20		Lecture only on Monday
111	160	2	Yes	Laboratory	2	20		
211	220	2	Yes	Laboratory	2	40	212	
212	260	2	Yes	Laboratory	2	30	211	
301	80	2	No	Laboratory	2	40		Tutorial must follow lecture
302	40	2	No	Laboratory	2	20		Lecture after 11 am
303	80	1	No	Laboratory	2	20		
304	40	1	No	Laboratory	2	40	305	Lectures to be scheduled only between 9.00am-2.00pm either on Tuesday or Thursday
305	40	1	No	Laboratory	2	30	304	
306	20	2	No	Laboratory	2	20		
308	50	1	No	Laboratory	1	20		
401	300	2	Yes	Laboratory	2	20		
402	200	2	No	Laboratory	2	20	403	
403	150	2	No	Laboratory	2	30	402	
404	100	2	No	Laboratory	2	20		
406	60	2	No	Laboratory	2	20		

Table 3.1 Sample data for twenty-four subjects with individual requirements

In solving the problem, we define each timetable item as an activity and the room as resource. Then we use the Scheduler to build the model of the CSP. Solver is needed to generate the goal and to search for a solution. When the solution is obtained, the timetable in term of subject number, time and room number are displayed. The pseudo-code for the solving timetabling problem is as follows:

```
Begin
    Use Scheduler to build the model;
    Define the variables of timetable item: subjects
    (activities), rooms (resources);
    Use Solver to generate the goal, and search for the
    solution;
    If (solver.solve(goal))
         Print the solution
    Else
         Print out "No solution"
End
```

Our system has three components: input, solver and output. In terms of input, individual lecturer will determine the requirements. This includes subject code, number of students enrolled, type of class requested (such as lecture or tutorial – classroom or laboratory), duration of lecture and tutorial, maximum number of students allowed to enrol in a tutorial group, requirement of having a repeat or second lecture and specifications of individual preferences or requirements which can be specified by individual lecturer. All input data will be stored in a data file.

In solving the problem, we design the model using the Scheduler. Firstly, we create the activities. Then the activity will be put into array and we create the unary resource of room for the activities. In our implementation we use Alternative Resource Set to represent a group of the resources. Finally, we create the temporal constraints and resource constraints. This is programmed as a

function and we use this function to get the model. The following pseudo-code shows the function on how to make the model in Scheduler.

```
Begin make model
    Define model: IloModel model (env);
    Create the activities:
      IloActivity activity (env, duration,subject_name);
    Put all the activities into array:
      IloActivityArray activities (env,NumberOfActivities);
    Create the resources:
      IloUnaryResource room (env,room_name);
    Put all the resources into array:
      IloUnaryResourceArray rooms (env, NumberOfRooms);
    Create the Alternative Resource Set:
      IloAltResSet altres (env,NumberOfRooms,rooms)
    Compare the room size to the student enrolled size
    If(room size> student size)
      Add the room to the alternative resource set
    Create the room resource constraints:
      Activities.requires(altres);
    Create the temporal constraints;
End make model
```

In this timetabling problem, we use `IloActivity` to represent the subjects which is represented as the timetable item of lecture and tutorial. We use `IloUnaryResource` to represent the room resource. An instance of `IloUnaryResource` represents a resource whose capacity is one. An instance

of `IloUnaryResource` is either occupied or free. No more than one activity can be executed at a given time because the room is unique to one subject in an allocated timeslot. If one subject occupies that timeslot in the room, other subjects cannot use that room at that timeslot. Rooms are grouped together into one alternative resource (`IloAlternativeResource`) with capacity greater than the number of student size. An alternative resource contains two or more resources with equivalent capabilities. An activity may require or provide only one resource from this set of alternative resources during its execution. Therefore, a subject only requires one single room from a set of alternative resources during its execution. In our program we have differentiated lecture and tutorial rooms separately.

We declare the variables and post the appropriate constraints in the program as discussed. The variables are used as activity to represent the subjects and `IloUnaryResource` to represent the rooms. Solver will propagates these constraints to reduce the domains of these variables. In Solver, the implementation of search algorithms is based on the idea of goals.

In terms of constraints, preferences listed by individual lecturer are expressed as temporal constraints in this case study. There constraints are listed as follows:

1. Subject S2 must start after Subject S1;

An example of this constraint is, for subject 315 the tutorial must be held after the students have attended the lecture.

```
Model.add(S2.startsAfterEnd (S1));
```

2. Different day (subject S1 must be scheduled on the different day as subjects S2)

   An example of this constraint is, for subject 214 there is two groups of lectures to be scheduled and the two lectures must be held on the different day.

```
Model.add(diffay(env, S1, S2));
```

3. Special day (subject S must be scheduled on a specific day D)

   An example of this constraint is, subject 307 must be scheduled only on Wednesday.

```
Model.add(S.endsBefore(D*11-1) &&
          S.startsAfter(D*11-11))
```

4. Non concurrent (subject S1 cannot be scheduled concurrently with subject S2)

   An example of this constraint is, subject 201 cannot be held concurrently with subjects 214, 311 and 312.

```
Model.add(nonconcurrent(S1, S2));
```

5. Reserve timeslot T (subject S cannot be scheduled at the timeslot T)

   For example, there should not be any lecture or tutorial scheduled on Wednesday 1:00 and 2:00pm due to university policy.

```
Model.add(S.startsAt(T));
```

6. Subject S must be scheduled before real time T

   For example, subject 311 must be scheduled before 2:00pm.

```
Model.add(S.endsBefore(T-8) && S.startsAfter(0)||
          S.endsBefore(T+3)  && S.startsAfter(11)||
          S.endsBefore(T+14) && S.startsAfter(22)||
          S.endsBefore(T+25) && S.startsAfter(33)||
          S.endsBefore(T+36) && S.startsAfter(44));
```

7. Subject S must be scheduled after timeslot T

   For example, subject 102 cannot be scheduled before 11am.

```
Model.add(S.startsAfter(T-8) && S.endsBefore(11)||
          S.startsAfter(T+3)  && S.endsBefore(22)||
          S.startsAfter(T+14) && S.endsBefore(33)||
          S.startsAfter(T+25) && S.endsBefore(44)||
          S.startsAfter(T+36) && S.endsBeofore(55));
```

Note that we have used number 0 to 54 to indicate specific day and time of the week. For example 0 is used to indicate Monday 8:00-9:00am, 1 is Monday 9:00-10:00am and so on.

3.4 Conclusion

In this chapter, we have described the sample case study problem and the application of ILOG Scheduler and Solver to model the problem. We will discuss the results obtained for the sample case study problem in the following chapter.

Chapter 4 Results

4.1 Introduction

In this chapter we will present and analyse the results obtained using different goals in ILOG Solver. The programs are run on a DELL personal computer with Intel Pentinum 4, 1.6 G CPU, 512 M memory and Linux 2.4 operating system.

ILOG Solver will solve the problem by first assigning values to variable in order to satisfy a given set of constraint. Then it enumerates values via branch and bound method by trying all values in domain and using general and custom searches through the application. Finally, constraint propagation is used to reduce domain at each node. These way inconsistent values are eliminated from current partial solution.

In ILOG, a goal is used to define the search for a solution to a model. The model for which an instance of a goal will search for a solution is specified via the `IloSolver`. That means once an `IloSolver` has extracted a model, a goal is associated with the model using the function `IloSolver::solve(const IloGoal)` or `IloSolver::startNewSearch(const IloGoal))`. In our experiment, we will use the predefined functions such as `IloRankForward`, `IloRankBackward`, `IloSetTimeForward` and `IloSetTimeBackward` that return a goal to assign start times to activities in a schedule. The program

written in ILOG is shown in Appendix 1. In addition we have added a goal `IloLimitSearch` in our program to set limits on the amount of time Solver spends on the search. When it reaches that time limit, Solver returns fail. For example, the `IloLimitSearch(env,goal,IloTimeLimit(env,120)` tells Solver to explore the search tree associated with the goal for only 120 seconds. Otherwise Solver returns no solution found.

To discuss the result of various goals for the sample case study problem, we will analyse the result from the perspectives of number of fails and number of choice points. Failure refers to the node which backtracks when the search cannot find the goal. The number of fails refers to the number of backtrack in the search process until a goal is found. Choice point refers to the node that has been explored or visited in the search process. Therefore the number of choice points refers to the number of nodes that has been visited in the search process. Figure 4.1 shows the Number of failures and Number of choice points in the search tree.

In figure 4.1, the grey circle represents choice point, and black circle represents dead end. The black square represents the goal is found, and the arrow represents the failure. For this four level search tree, there are seven choice points and three failures.

A choice point is created by the execution of the goal `IlcOr`. Backtracking occurs as long as no subgoal succeeds. Thus if no subgoal succeeds, the choice

point is considered as fail. We will also compare the result using the total running time that is explored in seconds. However, the time displays here returns the elapsed time, sometimes known as wall clock time.

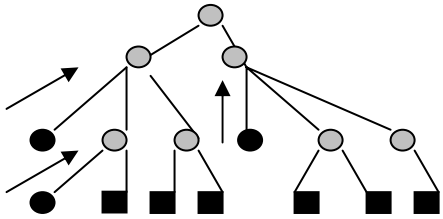


Figure 4.1 Number of failures and number of choice points

We have used the following various scenarios as a discussion basis in terms of the goals in the Solver. These include: ranking goals and SetTime goals. For each goal we have used various enforcement levels for each of the scenario.

The rest of chapter is organized as follow. Section 2 discusses the use of ranking goals to find the solution for the case study problem. Section 3 presents application of SetTime goals in solving the problem. `IloEnforcementLevel` which is to allow how much effort is specified in a given resource constraint is discussed in Section 4. Finally section 5 discusses the results and concludes the chapter.

4.2 Ranking Goals

The first scenario uses the ranking goals approach. As explained, ILOG uses the AC-5 algorithm that is the default search strategy in ILOG to remove inconsistent

values from variables domain. When a constraint is ranked first, the activity corresponding to it is positioned at the head of the activities not already ranked.

A set of instances of `IloResourceConstraint` may be ranked (ordered along the time line) for a resource. Ranking is defined for the classes `IloUnaryResource` and `IloStateResource.`The resource constraint selector selects the next resource constraint to be ranked first. A resource is chosen and the activities at each iteration, which require the chosen resource, are put in order. For this ordering at each iteration, a resource constraint is chosen (ILOG, 2001c).

4.2.1 IloRankForward

`IloRankForward` creates and returns a goal that ranks all resource constraints of the unary resource. By default (when no resource constraint selector is given as an argument), the resource constraint selector selects the next resource constraint to be ranked first.

The summary results obtained for the sample case study problem using this goal is as follows:

IloRankForward

```
Number of fails              : 373
Number of choice points      : 761
Number of variables          : 2159
Number of constraints        : 2988
Constraint queue (bytes)     : 9140
```

```
Total memory used (bytes)      : 2366756
Running time since creation    : 4.91
```

Here the number of constraints is the number of constraints extracted for invoking the Solver and number of variables is the number of constrained variables extracted for invoking the Solver. The detail timetable schedule for the sample case study problem is show in Appendix 2.

4.2.2 IloRankBackward

We use `IloRankBackward(env)` to rank all resource constrains. `IloRankBackward(const IloEnv env)` will return a goal that ranks all resource constraints of all unary resources in the model whose capacity constraints are not ignored. The difference with `IloRankForward` is that the next resource to be ranked backward by using `IloRankBackward`.

The summary results obtained for the sample case study problem using this goal is as follows:

IloRankBackward

```
Number of fails                : 380
Number of choice points        : 766
Number of variables            : 2159
Number of constraints          : 2983
Constraint queue (bytes)       : 9140
Total memory used (bytes)      : 2362736
Running time since creation    : 4.96
```

The detail timetable schedule for the sample case study problem is show in Appendix 3.

Discussion

The results show that two ranking goals have similar performance. From the detail timetable schedule, we can see the difference is that subjects were not held on the same time in different goals. For example, sub102 can start from Tuesday 11:00 -- 13:00 when using `IloRankForward.` However, the same subject (sub102) starts from Thursday 16:00 – 18:00 by using `IloRankBackward`. Because ranking goals only rank the constraints in different order, it will result in different timetable schedule.

4.3 SetTime Goal

The SetTime goal uses AC-5 and branch and bound algorithm to enable a minimum cost path to a goal to be found. Here we consider applying two goals: `IloSetTimeForward` and `IloSetTimeBackward.`

When we use the goal `IloSetTimesBackward`, the Solver will choose the latest timeslots for the activities (subjects) to build the timetable. On the contrary, the Solver will choose the start timeslots for the activities (subjects) when the goal `IloSetTimesForward` is selected.

4.3.1 IloSetTimesForward

Here we use the goal `IloSetTimesForward(env)` to find a solution. The `IloSetTimesForward` function creates and returns a goal that assigns a start

time to all activities in the model. The goal is designed to efficiently schedule activities in a chronological order. It considers only solutions that can be produced as follows (ILOG, 2001c). In each step, choose an unscheduled activity A of minimal earliest start time and schedule it as early as possible, as allowed by the previously scheduled activities (which have smaller start times than A).

Internally the function uses a schedule forward method that works by selecting activity to its earliest start time. If that start time leads to a failure, then the activity is postponed until its earliest start time has been removed. For example, suppose we have an activity B that can start as much as 100 units after the start of an activity A. This can be expressed by a precedence constraint with negative delay: `A.startsAfterStart(B, -100)`. Assume that B cannot be scheduled at a time earlier than 50 because it requires a resource that has a maximal capacity of 0 in the interval [0,50). Therefore A cannot be scheduled before time 100 even if there are no earlier activities. In this situation, `IlcSetTimes` will not find a solution here, because it will attempt to assign A to a start time of 0. When this fails, it will postpone A but then realize that there are no other activities that can be scheduled before A and therefore it concludes that there are no solutions. Note that in this very simple case, it is likely that constraint propagation will discover that 100 is the actual earliest start time of A and so `IlcSetTimes` will successfully find a solution. However, if this situation is part of a larger, more complex constraint interaction, it cannot be guaranteed that constraint propagation

will discover the globally consistent earliest start time. In such a situation, then a solution will be missed.

The summary result obtained using this goal for this case study problem is shown as follow:

IloSetTimeForward

```
Number of fails               : 2377
Number of choice points       : 2767
Number of variables           : 2159
Number of constraints         : 2998
Constraint queue (bytes)      : 9140
Total memory used (bytes)     : 2378816
Running time since creation   : 5.86
```

The detail timetable schedule for the sample case study problem is show in Appendix 4.

4.3.2 IloSetTimeBackward

The second case uses the goal `IloSetTimeBackward`. The `IloSetTimeBackward(env)` returns a goal that assigns an end time to each activity in the model. The function is designed to efficiently schedule activities in an anti-chronological order (ILOG, 2001c). It considers only solutions that can be produced as follows. In each step, an unscheduled activity A of maximal latest end time is selected and schedule it as late as possible as allowed by the previously scheduled activities (which have greater end times than A). Internally, the function uses the schedule backward method. Firstly the selected activity is

assigned to its latest end time. If that end time leads to a failure, then the activity is postponed backward until its latest end time has been removed.

As implied by the above description, IlcSetTimesBackward can be thought of as a "mirror image" of `IlcSetTimes`: rather than scheduling chronologically according to earliest start times and postponement, it schedules anti-chronologically according to latest end times and "backward" postponement. Consequently, there are also mirror image situations where IlcSetTimesBackward performs an incomplete search. For example, let's assume we have an activity B that can start a maximal 50 units after the start of an activity A which can be expressed by a precedence constraint with negative delay. A.startsAfterStart(B, -50). Given a scheduling horizon of 1000, suppose that A cannot be scheduled at a later time than 900 since it requires a resource that has a maximal capacity of 0 in the interval [900,1000). Then B cannot be scheduled after time 950 even if there are no later activities. In this case, `IlcSetTimesBackward` will not find a solution here, because it will attempt to assign B to an end time of 1000. When this fails, it will postpone-backward B but then realize that there are no other activities that can be schedule after B and therefore it concludes that there are no solutions. Note that in this very simple case, it is likely that constraint propagation will discover that 950 is the actual latest end time of B and so `IlcSetTimesBackward` will successfully find a solution. However, if this situation is part of a larger, more complex constraint

interaction, it cannot be guaranteed that constraint propagation will discover the globally consistent latest end time. In such a situation, then, a solution will be missed.

The summary result obtained using `IloSetTimeBackward` goal for the sample case study problem is shown as follow: The detail timetable schedule for the sample case study problem is show in Appendix 5.

IloSetTimeBackward

```
Number of fails               : 6460
Number of choice points       : 6856
Number of variables           : 2159
Number of constraints         : 2987
Constraint queue (bytes)      : 9140
Total memory used (bytes)     : 2382836
Running time since creation   : 8.72
```

Discussion

When the `IloSetTimeForward` goal is applied, the Solver puts the lecture at the earliest timeslots. In the sample case study, most of the tutorials are run after the lecture. When the temporal constraints such as tutorial must be scheduled after the lecture it is thus easy to satisfy. However for the `IloSetTimeBackward` goal, it will put the lecture at the latest timeslots. So the lecture needs to move backward to another timeslots when the program tries to satisfy the above constraints of scheduling tutorial after the lecture that will result in more failures than using the `IloSetTimeForward` goal.

From the results obtained above, we can see that using `IloSetTimeBackward` results in more number of failures and choice points than using `IloSetTimeForward`. The CPU time of these two scenarios only has a subtle difference due to small sample data. Obviously, the running time of using `IloSetTimeBackward` is more than using `IloSetTimeForward.` Due to a higher number of failures in `IloSetTimeBackward`, results obtained using this goal requires more CPU time.

4.4 Enforcement Level

Here we use `IloEnforcementLevel` to allow how much effort is specified in a given resource constraint. The enforcement level allows specifying with how much effort a given resource constraint may be expressed on a given resource. All levels ensure that any solution found by the scheduler will satisfy the type of constraint associated with the level. All levels ensure that any solution found by the scheduler will satisfy the type of constraint associated with the level. Stated otherwise, the enforcement level allows selecting which algorithms are used to enforce the corresponding constraint, but even the lowest enforcement level will ensure that the constraint is satisfied.

`IloBasic` is the default enforcement level. There are other enforcement levels that represent degrees of enforcement lower or higher than the `IloBasic`. Each level represents a certain degree of effort spent by the Scheduler to enforce

constraints. `IloMediumHigh`, `IloHigh` and `IloExtended` correspond to a scale of enforcement levels higher than the default level `IloBasic`. When the enforcement level of a type of constraint is higher than `IloBasic`, then the Scheduler will spend more effort at enforcing those constraints than it would by default. When higher enforcement levels is applied it causes more propagation of constraints, this results in fewer failures and fewer choice points, but more CPU time consumption in each search state. On the other hand, `IloLow` and `IloMediumLow` represent enforcement levels lower than the default level `IloBasic.` Thus Scheduler will spend less effort at enforcing those constraints than it would by default.

The higher enforcement levels typically cause more propagation of constraints; this results in fewer fails and fewer choice points, but more CPU time consumption in each search state. Also, the use of enforcement level should be chosen in accordance with the resource and how it is being used (ILOG, 2001b).

Here we will present summary results when different enforcement levels are combined with the ranking goals and SetTime goals. The summary results of each of the different scenarios are given as follows:

IloEnforceMentLevel: IloLow

IloRankForward:

```
Number of fails               : 373
```

```
Number of choice points      : 761
Number of variables          : 2159
Number of constraints        : 3000
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2541620
Running time since creation  : 5.23
```

IloRankBackward:

```
Number of fails              : 380
Number of choice points      : 766
Number of variables          : 2159
Number of constraints        : 2995
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2545640
Running time since creation  : 5.3
```

IloSetTimeForward:

```
Number of fails              : 2377
Number of choice points      : 2767
Number of variables          : 2159
Number of constraints        : 2998
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2378816
Running time since creation  : 5.87
```

IloSetTimeBackward:

```
Number of fails              : 6460
Number of choice points      : 6856
Number of variables          : 2159
Number of constraints        : 2987
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2382836
Running time since creation  : 8.72
```

IloEnforceMentLevel: IloHigh

IloRankForward:

```
Number of fails              : 3
Number of choice points      : 391
Number of variables          : 2159
Number of constraints        : 2988
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2274296
Running time since creation  : 3.91
```

IloRankBackward:

```
Number of fails              : 10
Number of choice points      : 396
Number of variables          : 2159
Number of constraints        : 2983
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2274296
Running time since creation  : 3.93
```

IloSetTimeForward:

```
Number of fails              : 2007
Number of choice points      : 2397
Number of variables          : 2159
Number of constraints        : 2998
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2314496
Running time since creation  : 5.25
```

IloSetTimeBackward

```
Number of fails              : 6090
Number of choice points      : 6486
Number of variables          : 2159
Number of constraints        : 2987
Constraint queue (bytes)     : 9140
Total memory used (bytes)    : 2314496
Running time since creation  : 9.33
```

IloEnforceMentLevel: IloExtended

IloRankForward:

```
Number of fails              : 3
Number of choice points      : 391
Number of variables          : 2159
Number of constraints        : 3024
Constraint queue (bytes)     : 11144
Total memory used (bytes)    : 39444944
Running time since creation  : 20.3
```

IloRankBackward:

```
Number of fails              : 10
Number of choice points      : 396
Number of variables          : 2159
Number of constraints        : 3019
Reversible stack (bytes)     : 14685084
Constraint queue (bytes)     : 11144
Total memory used (bytes)    : 40892144
Running time since creation  : 20.75
```

IloSetTimeForward:

```
Number of fails              : 2007
Number of choice points      : 2397
Number of variables          : 2159
Number of constraints        : 3034
Constraint queue (bytes)     : 11144
Total memory used (bytes)    : 39561524
Running time since creation  : 31.34
```

IloSetTimeBackward

```
Number of fails              : 6090
Number of choice points      : 6486
Number of variables          : 2159
Number of constraints        : 3023
Constraint queue (bytes)     : 11144
```

```
Total memory used (bytes)     : 40884104
Running time since creation   : 68.25
```

4.5 Conclusion

Table 4.1 summarise the various CPU time obtained as described in this chapter.

Scenarios	Number of fails	Number of choice points	CPU time (seconds)
1.IloRankForward	373	761	4.91
2.IloRankBackward	380	766	4.96
3.IloSetTimeForward	2377	2767	5.86
4.IloSetTimeBackward	6460	6856	8.72
5.IloLow+IloRankForward	373	761	5.23
6.IloLow+IloRankBackward	380	766	5.3
7.IloLow+IloSetTimeForward	2377	2767	5.87
8.IloLow+IloSetTimeBackward	6460	6856	8.72
9.IloHigh+IloRankForward	3	391	3.91
10.IloHigh+IloRankBackward	10	396	3.93
11.IloHigh+IloSetTimeForward	2007	2397	5.25
12.IloHigh+IloSetTimeBackforwad	6090	6486	9.33
13.IloExtended+IloRankForward	3	391	20.3
14.IloExtended +IloRankBackward	10	396	20.75
15.IloExtended +IloSetTimeForward	2007	2397	31.34
16.IloExtended+IloSetTimeBackward	6090	6486	68.25

Table 4.1 Comparisons of CPU time for different scenarios

We conclude that the scenario of `IloHigh+IloRankForward` gives the best result compare to other goals. In addition using the Ranking goals can get better result compare to using the SetTime goals.

Figure 4.2 shows the comparison of the number of fails and number of choice points between different scenarios.

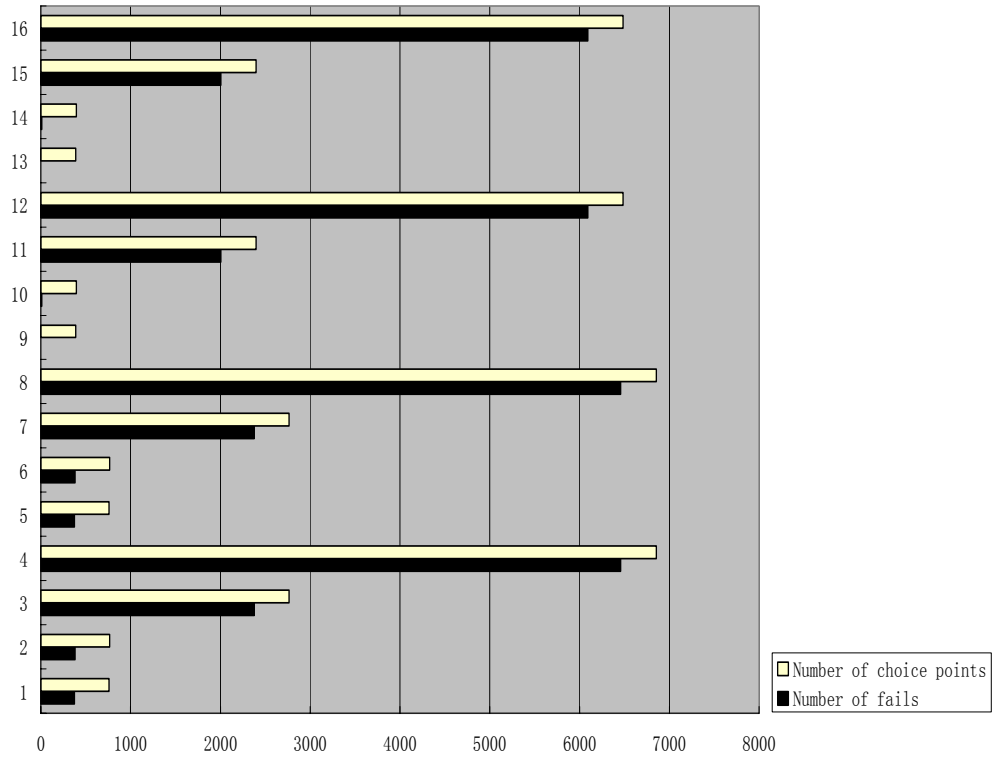


Figure 4.2 Comparison of the number of fails and number of choice points between different scenarios

Figure 4.3 shows the comparison of the CPU time between different scenarios.

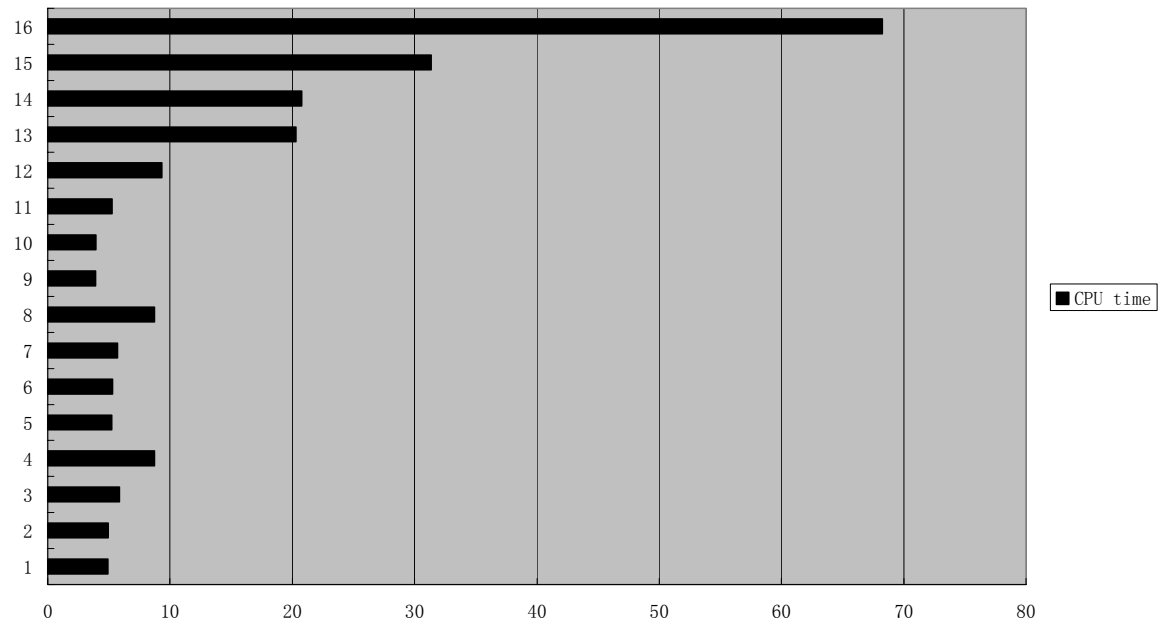


Figure 4.3 Comparison of the CPU time between different scenarios

Chapter 5 Conclusion

5.1 Overview of research

In this research, we have demonstrated that it is possible to apply the CSP approach to solve a university timetabling problem. The data for sample case study problem were derived from an information system department in the local university. We have used the CSP model to solve the problem. ILOG Solver and ILOG Scheduler tools are used to solve the CSP problem. Various scenarios in term of using different goals have been conducted and the results obtained are satisfactory.

5.2 Result of Research

1. Development of a timetabling problem using the CSP model.

   The CSP model consists of instantiation of the set of three tuples (timetable item, classroom, time). The timetable item is given by the set of subjects and implementation of sample case study has been developed using ILOG Scheduler and ILOG Solver.

2. Performance of different goals in ILOG.

   From the test results, we can see that using `IloHigh+IloRankForward` can lead to the better result compare to other goals. This means by enforcing

tight constraint level and ranking the constraints by positioning the constraints at the beginning of the activities can lead to better result. In addition we find that using Ranking goals can obtain better result compare to using SetTime goals.

5.3 Future work

There are two enhancements that can be made to the program. We propose future work to be conducted in the following areas:

1. ILOG Solver has its default search algorithm that is similar to AC-5. As a matter of conclusion, we will highlight that ILOG can make its own algorithm to search the solution. For future comparison, these algorithms can be implemented and compare to the results obtained here.

2. Currently the program just read the data from a data file. Future enhancement can be conducted to design a graphical user interface and a subject database connection. In addition the users can specify whether the preferences are of hard or soft constraints. This way when no optimal schedule is found, we can remove those preferences that are of soft constraints.

References

Abdennadher, S. and Marte, M. 2000, University course timetabling using Constraint Handling Rules, *Journal of Applied Artificial Intelligence*, vol.14, No.4, pp.311–326.

Abramson, D. and Abela, J., 1991, A Parallel Genetic Algorithm for Solving the School Timetabling Problem, *Technical Report*, Division of Information Technology, C.S.I.R.O.

Abramson, D., 1991, Constructing schools timetables using simulated annealing: sequential and parallel algorithms, *Management Science*, vol.1, No.37, pp.98-113.

Almond, M., 1966, An algorithm for constructing university timetables, *Computer Journal*, vol.8, pp.331-340.

Barták, R., 1998, *On-line Guide to Constraint Programming*, <http://kti.ms.mff.cuni.cz/~bartak/constraints/> (Access: 25 March, 2004).

Brailsford, S. C., Potts, C. N. and Smith, B. M., 1999, Constraint satisfaction problems: algorithms and applications, *European Journal of Operational Research*, vol.119, pp.557-581.

Brittan, J. N. G. and Farley, F. J. M., 1971, College timetable construction by computer, *The Computer Journal*, vol.14, pp.361–365.

Burke, E., Elliman, D. and Weare, R., 1994, A genetic algorithm based university timetabling system, *East-West Int. Conf. Computer Technologies in Education* vol.1, pp.35-40.

Busetti, F., 2003, *Simulated annealing overview,* <http://www.geocities.com/francorbusetti/saweb.pdf>, (Access: 25 July, 2004).

Chahal, N. and de Werra, D., 1989, An interactive system of constructing timetables on a PC, *European Journal of Operational Research*, vol.40, pp.32-37.

Chen, C. C. and Smith, S. F., 1997, Applying constraint satisfaction techniques to job shop scheduling, *Annals of Operations Research*, vol.70, pp.327-357.

Costa, D., 1994, A tabu search algorithm for computing an operational timetable, *European Journal of Operational Research*, vol.76, No.1, pp.98–110.

de Werra, D., 1985, An introduction to timetabling, *European Journal of Operations Research*, vol.19, pp.151–162.

de Werra, D., 1997, Restricted colouring models for timetabling, *Discrete Mathematics*, vol.165~166, pp.161–170.

Dechter, R., 2003, *Constraint Processing*, Morgan Kaufmann.

Deris, S. B., Omatu, S., Ohta, H. and Samat, P. A. B. D., 1997, University timetabling by constraint-base reasoning: A case study, *Journal of the Operational research Society*, vol.48, pp.1178-1190.

Duncan, A. K., 1965, Letters to the editor: Further results on a computer construction of school timetables, *Communications of the ACM*, vol.8, No.1, pp.72.

Even, S., Itai, A., and Shamir, A., 1976, On the complexity of timetable and multicommodity flow problems, *SIAM J. Computing*, vol.5, No.4, pp.691–703.

Franklin, T., Jenkins, E. and Woodson, K., 1995, A case study in scheduling courses, *UMAP Journal*, vol.15, no.2, pp.115-122.

Freuder, E., 1978, Synthesizing constraint expressions, *Communications of the ACM*, vol. 21, No.11, pp.958–966.

Gendrau, M., Laporte, G. and Potvin, J. Y., 1997, Vehicle routing: modern heuristics, In: Aarts, E. H. L. and Lenstra, J. K. (Eds.), *Local search in Combinatorial Optimizati*on, Wiley, Chichester, UK, pp.311-336.

Geske, U., 1998, Use of declarative programming for solving industrial optimization problems, <http://www.ercim.org/publication/Ercim_News/enw32/geske.html>, (Access: 25 March, 2004).

Glover, F., 1986, Future paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, vol3, pp.533-549.

Guéret, C., Jussien, N., Boizumault, P. and Prins, C., 1996, Building university timetables using constraint logic programming, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp. 130–145.

Gunadhi, H., Anand, V. J. and Yeong, W. Y., 1996, Automated timetabling using an object-oriented scheduler, *Expert systems with Applications*, vol.10, No. 2, pp.243-256.

Haralick, R. and Elliott, G., 1980, Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence*, vol.14, pp. 263-313.

Hertz, A., 1992, Finding a feasible course schedule using tabu search, *Discrete Applied Mathematics*, vol.35, pp.255-270.

Hertz, A., Taillard, E. and de Werra, D., 1997, A Tutorial on Tabu Search, In: Aarts, E. H. L. and Lenstra, J. K. (Eds.), *Local search in Combinatorial Optimizati*on, Wiley, Chichester, UK, pp.121–136.

IlOG, 2004, < http://www.ilog.com>, (Access: 25 July, 2004).

ILOG, Inc. 2001a, *ILOG Solver 5.1 User's Manual*, ILOG Inc.

ILOG, Inc. 2001b, *ILOG Scheduler 5.1 User's Manual*, ILOG Inc.

ILOG, Inc. 2001c, *ILOG Scheduler 5.1 Reference Manual*, ILOG Inc.

Jaffar, J. and Maher, M. J., 1994, Constraint logic programming: A survey, *The Journal of Logic Programming*, vol.19~20, pp.503-581.

Kumar, V., 1992, Algorithms for constraint Satisfaction Problems: A Survey, *AI magazine*, vol.13, No.1, pp.32-44.

Lajos, G., 1996, Complete university modular timetabling using constraint logic programming, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp. 146–161.

Mackworth, A. K. and Freuder, E., 1985, The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems, *Artificial Intelligence*, vol.25, pp.65-74.

Mackworth, A. K., 1977, Consistency in networks of relations, *Artificial Intelligence*, vol.8, No. 1, pp.99 – 118.

Montanari, U., 1974, Networks of constraints: Fundamental properties and application to picture processing, *Information Science*, vol.7, No.2, pp.95–132.

Mulvey, J., 1982, A classroom time assignment model, *European Journal of Operational Research*, vol.9, pp.64-70.

Nadel, B. A., 1989, Constraint Satisfaction Algorithms, *Computational Intelligence*, vol.5, pp.188-224.

O'Keefe, R., 1990, *The Craft of Prolog*, MIT Press., Cambridge, MA.

Ryan, D., 1992, Solution of massive generalized set partitioning problems in aircrew rostering. *Journal of the Operational Research Society*, vol.43, pp.459-467.

Sabin, D. and Freuder, E., 1994. Contradicting conventional wisdom in constraint satisfaction, In: Cohn, A.G. (Eds.), *Proceedings of European Conference on Artificial Intelligence (ECAI-94)*. Wiley, Chichester, UK, pp. 125-129.

Schaerf, A., 1999a, A survey of automated timetabling, *Artificial Intelligence Review*. vol.13, pp.87-127.

Schaerf, A., 1999b, Local search techniques for large high school timetabling problems, *IEEE Transactions on Systems Man and Cybernetics Part A – Systems and Humans*, vol.29, pp.368-377.

Terashima-Marín, H., 1998, Combinations of GAs and CSP Strategies for Solving the Examination Timetabling Problem, *PHD thesis.*

Tripathy, A., 1984, School Timetabling--A Case in Large Binary Integer Linear Programming, *Management Science*, vol.30, pp.1473–1489.

Tsang, E. P. K. 1993, *Foundations of Constraint Satisfaction*. Academic Press, Inc., San Diego, CA.

Van Hentenryck, P., 1989, *Constraint Satisfaction in Logic Programming*, MIT Press.

Van Hentenryck, P., Deville, Y., Teng, C. M., 1992, A generic arc-consistency algorithm and its specializations, *Artificial Intelligence*, vol.57, pp.291-321.

Vit'anyi, P. M. B., 1981, How well can a graph be n coloured, *Discrete mathematics*, vol.34, pp.69-80

Waltz, D., 1975, Understanding Line Drawings of Scenes with Shadows, *The Psychology of Computer Vision*, pp.19-91.

Wren, A., 1996, Scheduling, Timetabling and Rostering–A special Relationship, In: Burke, E. and Ross, P. (Eds.), *Practice and Theory of Automated Timetabling*, Springer-Verlag LNCS 1153, pp.46-75.

Appendix 1 - Program codes

```
/* TIMETABLE SYSTEM    */

#include<ilsched/iloscheduler.h>
#include<iomanip.h>
#include<string>
ILOSTLBEGIN

const IloInt NumberOfActivities=203;        //number of subjects
const IloInt NumberOfLectures=33;           //number of lectures in sample
                                            data(include repeat lectures)
const IloInt Lectures=24;                   //number of lectures    (the number of
                                            no repeat lectures)
const IloInt NumberOfTutorials=170;        //number of tutorials
const IloInt NumberOfRooms=4;              //number of lecture rooms
const IloInt NumberOfTutRooms=8;           //number of tutorial rooms
const IloNum horizon = 54;                  //number of timeslots

//read file exception class
class FileError : public IloException {
      public:
            FileError() : IloException("Cannot open data file") {}

};


////////////////////////////////////////////////////////////////////
//
//CONSTRAINTS    FUNCTIONS
//
////////////////////////////////////////////////////////////////////

//Subject S2 must start after Subject S1
IloConstraint afterSubject(IloActivity lec,IloActivity lec1)
{
      return(lec.startsAfterEnd(lec1));
}

// Subject S1 must be scheduled on the different day as Subjects S2
```

```
IloConstraint diffday(IloActivity lec, IloActivity lec1)
{
     return(lec.getStartVariable()-lec1.getStartVariable()>11);
}

//Subject S must be scheduled on a specific day D
IloConstraint specialday(IloActivity lec, IloInt D)
{

     return(lec.endsBefore(D*11-1) && lec.startsAfter(D*11-11));
}

//Non concurrent (subject S1 can not be scheduled concurrently with subject S2)
IloConstraint nonconcurrent(IloActivity lec, IloActivity lec1)
{
     return((lec.getStartVariable()+lec.getProcessingTimeVariable()
     <=lec1.getStartVariable())||(lec1.getStartVariable()+lec1.getProcessingTime
     Variable()<=lec.getStartVariable()));
}

//    Reserve Timeslot T (subject S can not be scheduled at the timeslot T)
IloConstraint reserve(IloActivity lec,IloInt T)
{
     return(lec.startsAt(T));
}

//    Subject S    must be scheduled before realtime T
IloConstraint beforeTime(IloActivity lec, IloInt T)
{

     return(lec.endsBefore(T-8)    && lec.startsAfter(0) ||
               lec.endsBefore(T+3)    && lec.startsAfter(11)||
               lec.endsBefore(T+14) && lec.startsAfter(22)||
               lec.endsBefore(T+25) && lec.startsAfter(33)||
               lec.endsBefore(T+36) && lec.startsAfter(44));
}

//   Subject S    must be scheduled After Timeslot T
IloConstraint afterTime(IloActivity lec, IloInt T)
{
     return( lec.startsAfter(T-8)    && lec.endsBefore(10)||
                lec.startsAfter(T+3)    && lec.endsBefore(21)||
                lec.startsAfter(T+14) && lec.endsBefore(32)||
```

```
                lec.startsAfter(T+25) && lec.endsBefore(43)||
                lec.startsAfter(T+36) && lec.endsBefore(54));
}

////////////////////////////////////////////////////////////////////
//
// READ DATA FROM FILE
//
////////////////////////////////////////////////////////////////////

void ReadFile(IloIntArray& dur, IloIntArray& tutdur,IloIntArray& studentsize,
                IloIntArray& tut_studentsize,     IloIntArray& roomsize,
                IloIntArray& tutroomsize,const char** RoomNames,const char**
                TutRoom,const char** LecNames, const char** TutNames)
{

    IloInt i;
    const char* fileName="./timetable.dat";
    ifstream file(fileName);
    if(!file)
        throw FileError();
    string Names[NumberOfLectures];
    string tutNames[NumberOfTutorials];
    string Rooms[NumberOfRooms];
    string tutRooms[NumberOfTutRooms];
    for(i=0;i<NumberOfLectures;i++)
        file>>dur[i];
    for(i=0;i<NumberOfTutorials;i++)
        file>>tutdur[i];
    for(i=0;i<NumberOfLectures;i++)
        file>>studentsize[i];
    for(i=0;i<NumberOfTutorials;i++)
        file>>tut_studentsize[i];
    for(i=0;i<NumberOfRooms;i++)
        file>>roomsize[i];
    for(i=0;i<NumberOfTutRooms;i++)
        file>>tutroomsize[i];
    for(i=0;i<NumberOfRooms;i++)
    {
        file>>Rooms[i];
        RoomNames[i]=Rooms[i].c_str();
    }
    for(i=0;i<NumberOfTutRooms;i++)
```

```
    {
        file>>tutRooms[i];
        TutRoom[i]=tutRooms[i].c_str();
    }
    for(i=0;i<NumberOfLectures;i++)
    {
        file>>Names[i];
        LecNames[i]=Names[i].c_str();
    }
    for(i=0;i<NumberOfTutorials;i++)
    {
        file>>tutNames[i];
        TutNames[i]=tutNames[i].c_str();
    }
}


////////////////////////////////////////////////////////////////////////////
//
//MAKE MODEL(USE SCHEDULER)
//
////////////////////////////////////////////////////////////////////////////

IloModel DefineModel(IloEnv env,IloEnforcementLevel level,IloNumVar&
                     makespan,IloIntArray dur, IloIntArray tutdur,IloIntArray
                     studentsize,IloIntArray tut_studentsize,IloIntArray
                     roomsize, IloIntArray tutroomsize,const char*
                     RoomNames[],const char* TutRoom[],    const char*
                     LecNames[], const char* TutNames[])
{
    IloInt i;
    IloModel model(env);
    IloSchedulerEnv schedEnv(env);
    makespan = IloNumVar(env, 0, horizon, IloNumVar::Int);

    //Set Capacity Enforcement
    //schedEnv.getResourceParam().setCapacityEnforcement(level);
    //Set Precedemce Enforcement
    schedEnv.getResourceParam().setPrecedenceEnforcement(level);

    /* DEFINE THE ACTIVITIES. */
    IloActivityArray activities(env,NumberOfActivities);
    for ( i = 0; i < NumberOfLectures; i++)
```

```
{
    activities[i]=IloActivity(env, dur[i],LecNames[i]);
    model.add(activities[i].endsBefore(horizon));
}
for(i=NumberOfLectures;i<NumberOfActivities;i++)
{
    activities[i]=IloActivity(env,tutdur[i-NumberOfLectures],TutNames[i-Nu
                    mberOfLectures]);
    model.add(activities[i].endsBefore(horizon));
}

/* DEFINE THE ROOM RESOURCE. */
IloUnaryResourceArray rooms(env,    NumberOfRooms);
for(IloInt k=0; k< NumberOfRooms;k++)
    rooms[k]=IloUnaryResource(env, RoomNames[k]);
IloUnaryResourceArray tutrooms(env, NumberOfTutRooms);
for(i=0;i<NumberOfTutRooms;i++)
    tutrooms[i]=IloUnaryResource(env,TutRoom[i]);

/* CREATE THE ALTERNATIVE RESOURCE SETS and CREATE
THE RESOURCE CONSTRAINTS. */
// resource constrains for lectures
for(IloInt j=0;j<NumberOfLectures;j++)
{
    IloAltResSet altres=IloAltResSet(env);
    for(i=0;i<NumberOfRooms;i++)
    {
        //compare the room size with the student size
        if(roomsize[i]>=studentsize[j])
            altres.add(rooms[i]);  //then add to alternative resource set
    }
    model.add(activities[j].requires(altres));
}
//resource constraints for tutorials
for(IloInt j=NumberOfLectures;j<NumberOfActivities;j++)
{
    IloAltResSet tut_altres=IloAltResSet(env);
    for(i=0;i<NumberOfTutRooms;i++)
    {
        if(tutroomsize[i]>=tut_studentsize[j-NumberOfLectures])
            tut_altres.add(tutrooms[i]);
    }
    model.add(activities[j].requires(tut_altres));
```

```
}
/* CREATE THE FAKE    ACTIVITY. */
IloActivityArray fakeactivities(env, 6);
for(i=0;i<6;i++)
      fakeactivities[i]=IloActivity(env, 1,"buffer");
// all the subjects can not hold    on timeslots=27
//all the lectures between 8:00am-6:00pm
model.add(reserve(fakeactivities[0],27));
model.add(reserve(fakeactivities[1],10));
model.add(reserve(fakeactivities[2],21));
model.add(reserve(fakeactivities[3],32));
model.add(reserve(fakeactivities[4],43));
model.add(reserve(fakeactivities[5],54));
for(i=0;i<6;i++)
{
      for(IloInt k=0; k< 4;k++)
            model.add(fakeactivities[i].requires(rooms[k]));
}
IloActivity rest=IloActivity(env,1,"buffer1");
// all the subjects can not hold on    timeslots=27
model.add(reserve(rest,27));
for(IloInt k=0; k<8;k++)
      model.add(rest.requires(tutrooms[k]));

/* CREATE THE TEMPORAL CONSTRAINTS. */
//lectures constraints
//sub102(not to have lecture before 11am)
model.add(afterTime(activities[0],11));

//sub110 , one lecture in the morning (11:00), and one in the afternoon(13:00)
model.add(beforeTime(activities[1],11));
model.add(afterTime(activities[2],13));
model.add(nonconcurrent(activities[1],activities[2]));

//sub201(not be held concurrently with sub214,sub311,sub312
model.add(nonconcurrent(activities[6],activities[3]));
model.add(nonconcurrent(activities[7],activities[3]));
model.add(nonconcurrent(activities[8],activities[3]));
model.add(nonconcurrent(activities[9],activities[3]));
model.add(nonconcurrent(activities[4],activities[3]));
model.add(nonconcurrent(activities[5],activities[3]));

//sub 214 , 2 lectures in different day
```

```
//not held concurrently with 201, 102
model.add(diffday(activities[4],activities[5]));
model.add(nonconcurrent(activities[0],activities[4]));
model.add(nonconcurrent(activities[0],activities[5]));

//sub 311, lecture only on 2,4
//not held concurrently with 312,315
model.add(nonconcurrent(activities[6],activities[7]));
model.add(nonconcurrent(activities[6],activities[8]));
model.add(nonconcurrent(activities[6],activities[9]));
model.add(nonconcurrent(activities[6],activities[10]));
model.add(nonconcurrent(activities[7],activities[8]));
model.add(nonconcurrent(activities[7],activities[9]));
model.add(nonconcurrent(activities[7],activities[10]));
model.add(specialday(activities[6],2)||specialday(activities[6],4));
model.add(specialday(activities[7],2)||specialday(activities[7],4));
model.add(nonconcurrent(activities[6],activities[7]));

//sub312 , lecture only on 1, 3, 5
// not held concurrently with 311
for(i=88;i<105;i++)
{
      model.add(nonconcurrent(activities[8],activities[i]));
      model.add(nonconcurrent(activities[9],activities[i]));
}
model.add(specialday(activities[8],1)||specialday(activities[8],3)||specialday(
                activities[8],5));
model.add(specialday(activities[9],1)||specialday(activities[9],3)||specialday(
                 activities[9],5));
model.add(nonconcurrent(activities[8],activities[9]));

//sub315, not held concurrently with 311,201,312,307
model.add(nonconcurrent(activities[8],activities[10]));
model.add(nonconcurrent(activities[9],activities[10]));
model.add(nonconcurrent(activities[3],activities[10]));
model.add(nonconcurrent(activities[11],activities[10]));

//sub307, only on wednesday
//not held concurrently with 311,312
model.add(nonconcurrent(activities[6],activities[11]));
model.add(nonconcurrent(activities[7],activities[11]));
model.add(nonconcurrent(activities[6],activities[11]));
model.add(nonconcurrent(activities[7],activities[11]));
```

```
model.add(specialday(activities[11],3));

//sub317 only on Monday
model.add(specialday(activities[12],1));
model.add(specialday(activities[13],1));
model.add(nonconcurrent(activities[12],activities[13]));

//sub211 not held concurently sub212
model.add(nonconcurrent(activities[16],activities[19]));
model.add(nonconcurrent(activities[16],activities[18]));
model.add(nonconcurrent(activities[16],activities[17]));
model.add(nonconcurrent(activities[17],activities[18]));
model.add(nonconcurrent(activities[17],activities[19]));

//sub302 lecture after 11am
model.add(afterTime(activities[21],11));

//sub304(Lectures to be scheduled only between 9.00am-2.00pm, on Tuesday
/   //or Thursday )
model.add(specialday(activities[23],2)||specialday(activities[23],4));
model.add(beforeTime(activities[23],14));
model.add(afterTime(activities[23],9));

//sub304 not concurrently with sub305
model.add(nonconcurrent(activities[23],activities[24]));

//sub402 not concurrently with sub403
model.add(nonconcurrent(activities[29],activities[30]));

//TUTORIAL
// tutorial102 after lecture102
for(i=33;i<38;i++)
    model.add(afterSubject(activities[i],activities[0]));
//tutorial 110
for(i=38;i<53;i++)
{
    model.add(nonconcurrent(activities[1],activities[i]));
    model.add(nonconcurrent(activities[2],activities[i]));
}
//tutorial 201 after lecture201
for(i=53;i<61;i++)
    model.add(afterSubject(activities[i],activities[3]));
//tutorial 214
```

```
for(i=61;i<73;i++)
{
    model.add(nonconcurrent(activities[4],activities[i]));
    model.add(nonconcurrent(activities[5],activities[i]));
}
//tutorial 311
for(i=73;i<88;i++)
{
    model.add(nonconcurrent(activities[i],activities[6]));
    model.add(nonconcurrent(activities[i],activities[7]));
}
//tutorial 312
for(i=88;i<105;i++)
{
    model.add(nonconcurrent(activities[8],activities[i]));
    model.add(nonconcurrent(activities[9],activities[i]));
}
//tutorial315 must be held after the lecture
for(i=105;i<110;i++)
    model.add(afterSubject(activities[i],activities[10]));
//tutorial 307
for(i=110;i<114;i++)
    model.add(nonconcurrent(activities[11],activities[i]));
//tutorial 317
for(i=114;i<127;i++)
{
    model.add(nonconcurrent(activities[i],activities[12]));
    model.add(nonconcurrent(activities[i],activities[13]));
    }
//tutorial 111
for(i=127;i<135;i++)
{
    model.add(nonconcurrent(activities[15],activities[i]));
    model.add(nonconcurrent(activities[15],activities[i]));
}
model.add(nonconcurrent(activities[14],activities[15]));
//tutorial211
for(i=135;i<141;i++)
{
    model.add(nonconcurrent(activities[16],activities[i]));
    model.add(nonconcurrent(activities[17],activities[i]));
}
//tutorial212
```

```
for(i=141;i<150;i++)
{
    model.add(nonconcurrent(activities[18],activities[i]));
    model.add(nonconcurrent(activities[19],activities[i]));
}
model.add(nonconcurrent(activities[18],activities[19]));

//tutorial301(tutorial after lecture)
for(i=150;i<152;i++)
{
    model.add(afterSubject(activities[i],activities[20]));
}
//tutorial 302
for(i=152;i<152;i++)
    model.add(nonconcurrent(activities[21],activities[i]));
//tutorial303
for(i=152;i<158;i++)
    model.add(nonconcurrent(activities[22],activities[i]));
//tutorial304
model.add(nonconcurrent(activities[23],activities[158]));
//tutorial 305
model.add(nonconcurrent(activities[24],activities[159]));
model.add(nonconcurrent(activities[24],activities[160]));
//tutorial 306
model.add(nonconcurrent(activities[25],activities[161]));
//tutorial 308
model.add(nonconcurrent(activities[26],activities[162]));
model.add(nonconcurrent(activities[26],activities[163]));
model.add(nonconcurrent(activities[26],activities[164]));
//tutorial401
for(i=165;i<180;i++)
{
    model.add(nonconcurrent(activities[27],activities[i]));
    model.add(nonconcurrent(activities[28],activities[i]));
}
//tutorial402
for(i=180;i<190;i++)
    model.add(nonconcurrent(activities[29],activities[i]));
//tutorial403
for(i=190;i<195;i++)
    model.add(nonconcurrent(activities[30],activities[i]));
//tutorial 404
for(i=195;i<200;i++)
```

```
        model.add(nonconcurrent(activities[31],activities[i]));
    //tutorial406
    for(i=200;i<203;i++)
    model.add(nonconcurrent(activities[32],activities[i]));

    /*RETURN THE MODEL*/
    return(model);
}


////////////////////////////////////////////////////////////////////////////////////////////
//
//CONVERT TIMESLOTS TO REAL TIME FOR PRINTING
//
////////////////////////////////////////////////////////////////////////////////////////////

void digit2time(IlcInt starttimeslots,IlcInt endtimeslots)
{

    cout<<" from\t ";
    switch(starttimeslots/11)
    {
        case 0:
            cout<<"Mon.";
            break;
        case 1:
            cout<<"Tue.";
            break;
        case 2:
            cout<<"Wed.";
            break;
        case 3:
            cout<<"Thu.";
            break;
        case 4:
            cout<<"Fri.";
            break;
    }
    cout<<"\t";
    switch((starttimeslots+11)%11)
    {
        case 0:
            cout<<" 8:00";
```

```
            break;
        case 1:
            cout<<" 9:00";
            break;
        case 2:
            cout<<" 10:00";
            break;
        case 3:
            cout<<" 11:00";
            break;
        case 4:
            cout<<" 12:00";
            break;
        case 5:
            cout<<" 13:00";
            break;
        case 6:
            cout<<" 14:00";
            break;
        case 7:
            cout<<" 15:00";
            break;
        case 8:
            cout<<" 16:00";
            break;
        case 9:
            cout<<" 17:00";
            break;
        case 10:
            cout<<" 18:00";
            break;

 }
 cout<<"\tto ";
switch((endtimeslots+11)%11)
{
        case 0:
            cout<<" 19:00";
            break;
        case 1:
            if(starttimeslots/11==endtimeslots/11)
                cout<<"    9:00";
            else
```

```
                cout<<" 20:00";
            break;
        case 2:
            cout<<" 10:00";
            break;
        case 3:
            cout<<" 11:00";
            break;
        case 4:
            cout<<" 12:00";
            break;
        case 5:
            cout<<" 13:00";
            break;
        case 6:
            cout<<" 14:00";
            break;
        case 7:
            cout<<" 15:00";
            break;
        case 8:
            cout<<" 16:00";
            break;
        case 9:
            cout<<" 17:00";
            break;
        case 10:
            cout<<" 18:00";
            break;

    }


}

////////////////////////////////////////////////////////////////////////////
//
// PRINTING OF SOLUTIONS
//
////////////////////////////////////////////////////////////////////////////

void PrintSolution(const IloSolver solver)
{
```

```
    IlcScheduler scheduler(solver);
    IloEnv env = solver.getEnv();
     solver.out()<<endl<<"-------------------------TIMETABLE
                                SOLUTION"<<"-------------------------\n\n";
    for(IloIterator<IloResourceConstraint> rctIter(env);rctIter.ok(); ++rctIter)
    {
        IloResourceConstraint rct = *rctIter;
        if (scheduler.hasAlternative(rct))
        {
            IlcAltResConstraint ilcAltRct = scheduler.getAltResConstraint(rct);
            env.out() << setiosflags(ios::left)<<setw(10)<<
                        ilcAltRct.getActivity().getName()<<setw(6)<<" ";
                        digit2time(ilcAltRct.getActivity().getStartMin(),ilcAltR
                        ct.getActivity().getEndMin());
            env.out()<<"\tuses "<<ilcAltRct.getSelected().getName()<<endl ;
        }
    }
//PRINT THE SOLVER INFORMATIN , SUCH AS : NUMBER OF
//FAILS,NUMBER OF CHOICE POINT
    solver.printInformation();
}


////////////////////////////////////////////////////////////////////////////////////////
//
// MAIN FUNCTION (USE SOLVER TO GET SOLUTION)
//
////////////////////////////////////////////////////////////////////////////////////////
int main(int argc, char** argv)
{
    try {

            IloEnv env;
            IloInt i;
            IloIntArray dur(env,NumberOfLectures);    //Lectures duration time
            IloIntArray tutdur(env,NumberOfTutorials);//Tutorials duration time
            IloIntArray studentsize(env,NumberOfLectures);//Number of student
            IloIntArray roomsize(env,NumberOfRooms);    //Lecture room size
            IloIntArray tutroomsize(env,NumberOfTutRooms); //Tutorial room
                                                            size
            //Maximum number of students in a tutorial
            IloIntArray tut_studentsize(env,NumberOfTutorials);
            const char* LecNames[NumberOfLectures];      //Name of lectures
```

```
const char* TutNames[NumberOfTutorials];     //Name of tutorials
const char* RoomNames[NumberOfRooms];    //Name of lecture
                                                    rooms
const char* TutRoom[NumberOfTutRooms];   //Name of tutorial
                                                     rooms

IloNumVar    makespan;
//Enforcement level
IloEnforcementLevel level = IloBasic;
if (argc > 1) {
      if (!strcmp(argv[1], "IloLow"))
            level = IloLow;
      else if (!strcmp(argv[1], "IloBasic"))
            level = IloBasic;
      else if (!strcmp(argv[1], "IloHigh"))
            level = IloHigh;
      else if (!strcmp(argv[1], "IloExtended"))
            level = IloExtended;
}
ReadFile(dur,tutdur,studentsize,tut_studentsize,roomsize,tutroomsize
,RoomNames,TutRoom,LecNames,TutNames);

IloModel model=DefineModel(env,level,makespan,
                                    dur,tutdur,studentsize,
                                    tut_studentsize,roomsize,
                                    tutroomsize,RoomNames,
                              TutRoom,LecNames,TutNames);

//Algorithm
IloSolver solver(model);
//SCENARIO 1: Raking goals
//IloGoal goal=IloAssignAlternative(env)&& IloRankForward(env);
IloGoal goal=IloAssignAlternative(env)&& IloRankBackward(env);

//SCENARIO 2: SetTime
//IloGoal goal = IloAssignAlternative(env) &&
IloSetTimesForward(env);
//IloGoal goal = IloAssignAlternative(env) &&
IloSetTimesBackward(env);
if(solver.solve(goal &&
IloLimitSearch(env,goal,IloTimeLimit(env,120))))
      PrintSolution(solver);
else
```

```
                    solver.out()<<"No solution"<<endl;
                env.end();
            }catch(IloException& exc){
            cout<<exc<<endl;
        }
        return 0;
}

////////////////////////////////////////////////////////////////////////////
//
// END PROGRAM
//
////////////////////////////////////////////////////////////////////////////
```

Appendix 2 - Detail timetable using IloRankForward goal

```
----------------------TIMETABLE SOLUTION-------------------------

sub102          from   Tue.   11:00 to  13:00  uses Room1A
sub110A         from   Wed.   8:00  to  10:00  uses Room4A
sub110B         from   Tue.   13:00 to  15:00  uses Room4A
sub201          from   Mon.   14:00 to  16:00  uses Room2A
sub214A         from   Tue.   13:00 to  15:00  uses Room1A
sub214B         from   Mon.   12:00 to  14:00  uses Room1A
sub311A         from   Thu.   11:00 to  13:00  uses Room2A
sub311B         from   Thu.   13:00 to  15:00  uses Room2A
sub312A         from   Wed.   16:00 to  18:00  uses Room3A
sub312B         from   Fri.   8:00  to  10:00  uses Room3A
sub315          from   Mon.   8:00  to  10:00  uses Room1A
sub307          from   Wed.   8:00  to   9:00  uses Room1A
sub317A         from   Mon.   8:00  to  10:00  uses Room2A
sub317B         from   Mon.   10:00 to  12:00  uses Room2A
sub111A         from   Tue.   16:00 to  18:00  uses Room1A
sub111B         from   Mon.   14:00 to  16:00  uses Room1A
sub211A         from   Wed.   9:00  to  11:00  uses Room1A
sub211B         from   Wed.   11:00 to  13:00  uses Room1A
sub212A         from   Mon.   12:00 to  14:00  uses Room2A
sub212B         from   Tue.   8:00  to  10:00  uses Room2A
sub301          from   Mon.   10:00 to  12:00  uses Room1A
sub302          from   Wed.   14:00 to  16:00  uses Room1A
sub303          from   Thu.   11:00 to  12:00  uses Room1A
sub304          from   Thu.   10:00 to  11:00  uses Room1A
sub305          from   Tue.   15:00 to  16:00  uses Room1A
sub306          from   Wed.   16:00 to  18:00  uses Room1A
sub308          from   Thu.   12:00 to  13:00  uses Room1A
sub401A         from   Mon.   16:00 to  18:00  uses Room2A
sub401B         from   Tue.   10:00 to  12:00  uses Room2A
sub402          from   Mon.   8:00  to  10:00  uses Room3A
sub403          from   Tue.   12:00 to  14:00  uses Room2A
sub404          from   Thu.   8:00  to  10:00  uses Room1A
sub406          from   Mon.   16:00 to  18:00  uses Room1A
tut102a         from   Tue.   13:00 to  15:00  uses Room17
```

```
tut102b        from   Tue.   15:00 to  17:00  uses Room17
tut102c        from   Tue.   17:00 to  19:00  uses Room17
tut102d        from   Wed.   8:00  to  10:00  uses Room17
tut102e        from   Wed.   10:00 to  12:00  uses Room17
tut110a        from   Mon.   8:00  to  10:00  uses Room18
tut110b        from   Mon.   10:00 to  12:00  uses Room18
tut110c        from   Mon.   12:00 to  14:00  uses Room18
tut110d        from   Mon.   14:00 to  16:00  uses Room18
tut110e        from   Mon.   16:00 to  18:00  uses Room18
tut110f        from   Mon.   18:00 to  20:00  uses Room18
tut110g        from   Tue.   9:00  to  11:00  uses Room18
tut110h        from   Mon.   8:00  to  10:00  uses Room16
tut110i        from   Mon.   10:00 to  12:00  uses Room16
tut110j        from   Mon.   12:00 to  14:00  uses Room16
tut110k        from   Mon.   14:00 to  16:00  uses Room16
tut110l        from   Mon.   16:00 to  18:00  uses Room16
tut110m        from   Mon.   18:00 to  20:00  uses Room16
tut110n        from   Tue.   9:00  to  11:00  uses Room16
tut110o        from   Tue.   11:00 to  13:00  uses Room16
tut201a        from   Tue.   13:00 to  15:00  uses Room16
tut201b        from   Tue.   15:00 to  17:00  uses Room16
tut201c        from   Tue.   17:00 to  19:00  uses Room16
tut201d        from   Wed.   8:00  to  10:00  uses Room16
tut201e        from   Wed.   10:00 to  12:00  uses Room16
tut201f        from   Wed.   14:00 to  16:00  uses Room16
tut201g        from   Wed.   16:00 to  18:00  uses Room16
tut201h        from   Wed.   18:00 to  20:00  uses Room16
tut214a        from   Thu.   9:00  to  11:00  uses Room16
tut214b        from   Thu.   11:00 to  13:00  uses Room16
tut214c        from   Thu.   13:00 to  15:00  uses Room16
tut214d        from   Thu.   15:00 to  17:00  uses Room16
tut214e        from   Thu.   17:00 to  19:00  uses Room16
tut214f        from   Fri.   8:00  to  10:00  uses Room16
tut214g        from   Fri.   10:00 to  12:00  uses Room16
tut214h        from   Fri.   12:00 to  14:00  uses Room16
tut214i        from   Fri.   14:00 to  16:00  uses Room16
tut214j        from   Fri.   16:00 to  18:00  uses Room16
tut214k        from   Mon.   8:00  to  10:00  uses Room15
tut214m        from   Mon.   10:00 to  12:00  uses Room15
tut311a        from   Mon.   12:00 to  14:00  uses Room15
tut311b        from   Mon.   14:00 to  16:00  uses Room15
tut311c        from   Mon.   16:00 to  18:00  uses Room15
tut311d        from   Mon.   18:00 to  20:00  uses Room15
```

```
tut311e        from   Tue.   9:00  to  11:00  uses Room15
tut311f        from   Tue.   11:00 to  13:00  uses Room15
tut311g        from   Tue.   13:00 to  15:00  uses Room15
tut311h        from   Tue.   15:00 to  17:00  uses Room15
tut311i        from   Tue.   17:00 to  19:00  uses Room15
tut311j        from   Wed.   8:00  to  10:00  uses Room15
tut311k        from   Wed.   10:00 to  12:00  uses Room15
tut311l        from   Wed.   14:00 to  16:00  uses Room15
tut311m        from   Wed.   16:00 to  18:00  uses Room15
tut311n        from   Wed.   18:00 to  20:00  uses Room15
tut311o        from   Thu.   9:00  to  11:00  uses Room15
tut312a        from   Thu.   11:00 to  13:00  uses Room15
tut312b        from   Mon.   8:00  to  10:00  uses Room14
tut312c        from   Mon.   10:00 to  12:00  uses Room14
tut312d        from   Mon.   12:00 to  14:00  uses Room14
tut312e        from   Mon.   14:00 to  16:00  uses Room14
tut312f        from   Mon.   16:00 to  18:00  uses Room14
tut312g        from   Mon.   18:00 to  20:00  uses Room14
tut312h        from   Tue.   9:00  to  11:00  uses Room14
tut312i        from   Tue.   11:00 to  13:00  uses Room14
tut312j        from   Tue.   13:00 to  15:00  uses Room14
tut312k        from   Tue.   15:00 to  17:00  uses Room14
tut312l        from   Tue.   17:00 to  19:00  uses Room14
tut312m        from   Wed.   8:00  to  10:00  uses Room14
tut312n        from   Wed.   10:00 to  12:00  uses Room14
tut312o        from   Wed.   14:00 to  16:00  uses Room14
tut312p        from   Mon.   8:00  to  10:00  uses Room13
tut312q        from   Mon.   10:00 to  12:00  uses Room13
tut315a        from   Mon.   12:00 to  14:00  uses Room13
tut315b        from   Mon.   14:00 to  16:00  uses Room13
tut315c        from   Mon.   16:00 to  18:00  uses Room13
tut315d        from   Mon.   18:00 to  20:00  uses Room13
tut315e        from   Tue.   9:00  to  11:00  uses Room13
tut307a        from   Wed.   16:00 to  18:00  uses Room14
tut307b        from   Wed.   18:00 to  20:00  uses Room14
tut307c        from   Thu.   9:00  to  11:00  uses Room14
tut307d        from   Tue.   11:00 to  13:00  uses Room13
tut317a        from   Tue.   13:00 to  15:00  uses Room13
tut317b        from   Tue.   15:00 to  17:00  uses Room13
tut317c        from   Tue.   17:00 to  19:00  uses Room13
tut317d        from   Wed.   8:00  to  10:00  uses Room13
tut317e        from   Wed.   10:00 to  12:00  uses Room13
tut317f        from   Wed.   14:00 to  16:00  uses Room13
```

```
tut317g        from   Wed.   16:00 to  18:00  uses Room13
tut317h        from   Wed.   18:00 to  20:00  uses Room13
tut317i        from   Thu.   9:00  to  11:00  uses Room13
tut317j        from   Thu.   11:00 to  13:00  uses Room13
tut317k        from   Thu.   13:00 to  15:00  uses Room13
tut317l        from   Thu.   15:00 to  17:00  uses Room13
tut317m        from   Thu.   17:00 to  19:00  uses Room13
tut111a        from   Fri.   8:00  to  10:00  uses Room13
tut111b        from   Fri.   10:00 to  12:00  uses Room13
tut111c        from   Fri.   12:00 to  14:00  uses Room13
tut111d        from   Fri.   14:00 to  16:00  uses Room13
tut111e        from   Fri.   16:00 to  18:00  uses Room13
tut111f        from   Mon.   8:00  to  10:00  uses Room12
tut111g        from   Mon.   10:00 to  12:00  uses Room12
tut111h        from   Mon.   12:00 to  14:00  uses Room12
tut211a        from   Thu.   13:00 to  15:00  uses Room15
tut211b        from   Thu.   15:00 to  17:00  uses Room15
tut211c        from   Thu.   17:00 to  19:00  uses Room15
tut211d        from   Fri.   8:00  to  10:00  uses Room15
tut211e        from   Fri.   10:00 to  12:00  uses Room15
tut211f        from   Mon.   14:00 to  16:00  uses Room12
tut212a        from   Thu.   11:00 to  13:00  uses Room14
tut212b        from   Thu.   13:00 to  15:00  uses Room14
tut212c        from   Thu.   15:00 to  17:00  uses Room14
tut212d        from   Thu.   17:00 to  19:00  uses Room14
tut212e        from   Fri.   8:00  to  10:00  uses Room14
tut212f        from   Fri.   10:00 to  12:00  uses Room14
tut212g        from   Fri.   12:00 to  14:00  uses Room14
tut212h        from   Fri.   14:00 to  16:00  uses Room14
tut212i        from   Mon.   16:00 to  18:00  uses Room12
tut301a        from   Fri.   12:00 to  14:00  uses Room15
tut301b        from   Fri.   14:00 to  16:00  uses Room15
tut302a        from   Mon.   18:00 to  20:00  uses Room12
tut302b        from   Tue.   9:00  to  11:00  uses Room12
tut303a        from   Tue.   11:00 to  13:00  uses Room12
tut303b        from   Tue.   13:00 to  15:00  uses Room12
tut303c        from   Tue.   15:00 to  17:00  uses Room12
tut303d        from   Tue.   17:00 to  19:00  uses Room12
tut304a        from   Fri.   16:00 to  18:00  uses Room15
tut305a        from   Fri.   16:00 to  18:00  uses Room14
tut305b        from   Wed.   8:00  to  10:00  uses Room12
tut306a        from   Wed.   10:00 to  12:00  uses Room12
tut308a        from   Wed.   12:00 to  13:00  uses Room12
```

```
tut308b        from   Fri.   16:00 to  17:00  uses Room12
tut308c        from   Wed.   12:00 to  13:00  uses Room11
tut401a        from   Wed.   14:00 to  16:00  uses Room12
tut401b        from   Wed.   16:00 to  18:00  uses Room12
tut401c        from   Wed.   18:00 to  20:00  uses Room12
tut401d        from   Thu.   9:00  to  11:00  uses Room12
tut401e        from   Thu.   11:00 to  13:00  uses Room12
tut401f        from   Thu.   13:00 to  15:00  uses Room12
tut401g        from   Thu.   15:00 to  17:00  uses Room12
tut401h        from   Thu.   17:00 to  19:00  uses Room12
tut401i        from   Fri.   8:00  to  10:00  uses Room12
tut401j        from   Fri.   10:00 to  12:00  uses Room12
tut401k        from   Fri.   12:00 to  14:00  uses Room12
tut401l        from   Fri.   14:00 to  16:00  uses Room12
tut401m        from   Mon.   8:00  to  10:00  uses Room11
tut401n        from   Mon.   10:00 to  12:00  uses Room11
tut401o        from   Mon.   12:00 to  14:00  uses Room11
tut402a        from   Mon.   14:00 to  16:00  uses Room11
tut402b        from   Mon.   16:00 to  18:00  uses Room11
tut402c        from   Mon.   18:00 to  20:00  uses Room11
tut402d        from   Tue.   9:00  to  11:00  uses Room11
tut402e        from   Tue.   11:00 to  13:00  uses Room11
tut402f        from   Tue.   13:00 to  15:00  uses Room11
tut402g        from   Tue.   15:00 to  17:00  uses Room11
tut402h        from   Tue.   17:00 to  19:00  uses Room11
tut402i        from   Wed.   8:00  to  10:00  uses Room11
tut402j        from   Wed.   10:00 to  12:00  uses Room11
tut403a        from   Wed.   14:00 to  16:00  uses Room11
tut403b        from   Wed.   16:00 to  18:00  uses Room11
tut403c        from   Wed.   18:00 to  20:00  uses Room11
tut403d        from   Thu.   9:00  to  11:00  uses Room11
tut403e        from   Thu.   11:00 to  13:00  uses Room11
tut404a        from   Thu.   13:00 to  15:00  uses Room11
tut404b        from   Thu.   15:00 to  17:00  uses Room11
tut404c        from   Thu.   17:00 to  19:00  uses Room11
tut404d        from   Fri.   8:00  to  10:00  uses Room11
tut404e        from   Fri.   10:00 to  12:00  uses Room11
tut406a        from   Fri.   12:00 to  14:00  uses Room11
tut406b        from   Fri.   14:00 to  16:00  uses Room11
tut406c        from   Fri.   16:00 to  18:00  uses Room11
```

Appendix 3 - Detail timetable using IloRankBackward goal

```
----------------------TIMETABLE SOLUTION-------------------------

sub102          from   Thu.   16:00 to  18:00  uses Room1A
sub110A         from   Fri.   8:00  to  10:00  uses Room4A
sub110B         from   Fri.   13:00 to  15:00  uses Room4A
sub201          from   Thu.   8:00  to  10:00  uses Room2A
sub214A         from   Fri.   16:00 to  18:00  uses Room1A
sub214B         from   Thu.   14:00 to  16:00  uses Room1A
sub311A         from   Tue.   10:00 to  12:00  uses Room2A
sub311B         from   Tue.   8:00  to  10:00  uses Room2A
sub312A         from   Wed.   8:00  to  10:00  uses Room3A
sub312B         from   Mon.   8:00  to  10:00  uses Room3A
sub315          from   Thu.   12:00 to  14:00  uses Room1A
sub307          from   Wed.   17:00 to  18:00  uses Room1A
sub317A         from   Mon.   10:00 to  12:00  uses Room2A
sub317B         from   Mon.   8:00  to  10:00  uses Room2A
sub111A         from   Fri.   12:00 to  14:00  uses Room1A
sub111B         from   Fri.   10:00 to  12:00  uses Room1A
sub211A         from   Fri.   8:00  to  10:00  uses Room1A
sub211B         from   Thu.   10:00 to  12:00  uses Room1A
sub212A         from   Fri.   16:00 to  18:00  uses Room2A
sub212B         from   Fri.   14:00 to  16:00  uses Room2A
sub301          from   Thu.   8:00  to  10:00  uses Room1A
sub302          from   Wed.   15:00 to  17:00  uses Room1A
sub303          from   Wed.   8:00  to   9:00  uses Room1A
sub304          from   Tue.   9:00  to  10:00  uses Room1A
sub305          from   Tue.   10:00 to  11:00  uses Room1A
sub306          from   Wed.   11:00 to  13:00  uses Room1A
sub308          from   Wed.   14:00 to  15:00  uses Room1A
sub401A         from   Fri.   10:00 to  12:00  uses Room2A
sub401B         from   Fri.   8:00  to  10:00  uses Room2A
sub402          from   Fri.   14:00 to  16:00  uses Room3A
sub403          from   Fri.   12:00 to  14:00  uses Room2A
sub404          from   Wed.   9:00  to  11:00  uses Room1A
sub406          from   Fri.   14:00 to  16:00  uses Room1A
tut102a         from   Fri.   15:00 to  17:00  uses Room17
```

```
tut102b        from   Fri.   13:00 to  15:00  uses Room17
tut102c        from   Fri.   11:00 to  13:00  uses Room17
tut102d        from   Fri.   9:00  to  11:00  uses Room17
tut102e        from   Thu.   18:00 to  20:00  uses Room17
tut110a        from   Thu.   15:00 to  17:00  uses Room18
tut110b        from   Thu.   13:00 to  15:00  uses Room18
tut110c        from   Thu.   11:00 to  13:00  uses Room18
tut110d        from   Thu.   9:00  to  11:00  uses Room18
tut110e        from   Wed.   18:00 to  20:00  uses Room18
tut110f        from   Wed.   16:00 to  18:00  uses Room18
tut110g        from   Wed.   14:00 to  16:00  uses Room18
tut110h        from   Thu.   11:00 to  13:00  uses Room16
tut110i        from   Thu.   9:00  to  11:00  uses Room16
tut110j        from   Wed.   18:00 to  20:00  uses Room16
tut110k        from   Wed.   16:00 to  18:00  uses Room16
tut110l        from   Wed.   14:00 to  16:00  uses Room16
tut110m        from   Wed.   10:00 to  12:00  uses Room16
tut110n        from   Wed.   8:00  to  10:00  uses Room16
tut110o        from   Tue.   17:00 to  19:00  uses Room16
tut201a        from   Fri.   16:00 to  18:00  uses Room16
tut201b        from   Fri.   14:00 to  16:00  uses Room16
tut201c        from   Fri.   12:00 to  14:00  uses Room16
tut201d        from   Fri.   10:00 to  12:00  uses Room16
tut201e        from   Fri.   8:00  to  10:00  uses Room16
tut201f        from   Thu.   17:00 to  19:00  uses Room16
tut201g        from   Thu.   15:00 to  17:00  uses Room16
tut201h        from   Thu.   13:00 to  15:00  uses Room16
tut214a        from   Tue.   15:00 to  17:00  uses Room16
tut214b        from   Tue.   13:00 to  15:00  uses Room16
tut214c        from   Tue.   11:00 to  13:00  uses Room16
tut214d        from   Tue.   9:00  to  11:00  uses Room16
tut214e        from   Mon.   18:00 to  20:00  uses Room16
tut214f        from   Mon.   16:00 to  18:00  uses Room16
tut214g        from   Mon.   14:00 to  16:00  uses Room16
tut214h        from   Mon.   12:00 to  14:00  uses Room16
tut214i        from   Mon.   10:00 to  12:00  uses Room16
tut214j        from   Mon.   8:00  to  10:00  uses Room16
tut214k        from   Fri.   12:00 to  14:00  uses Room15
tut214m        from   Fri.   10:00 to  12:00  uses Room15
tut311a        from   Fri.   8:00  to  10:00  uses Room15
tut311b        from   Thu.   17:00 to  19:00  uses Room15
tut311c        from   Thu.   15:00 to  17:00  uses Room15
tut311d        from   Thu.   13:00 to  15:00  uses Room15
```

```
tut311e        from   Thu.   11:00 to  13:00  uses Room15
tut311f        from   Thu.   9:00  to  11:00  uses Room15
tut311g        from   Wed.   18:00 to  20:00  uses Room15
tut311h        from   Wed.   16:00 to  18:00  uses Room15
tut311i        from   Wed.   14:00 to  16:00  uses Room15
tut311j        from   Wed.   11:00 to  13:00  uses Room15
tut311k        from   Wed.   9:00  to  11:00  uses Room15
tut311l        from   Tue.   18:00 to  20:00  uses Room15
tut311m        from   Tue.   16:00 to  18:00  uses Room15
tut311n        from   Tue.   14:00 to  16:00  uses Room15
tut311o        from   Tue.   12:00 to  14:00  uses Room15
tut312a        from   Tue.   9:00  to  11:00  uses Room15
tut312b        from   Fri.   16:00 to  18:00  uses Room14
tut312c        from   Fri.   14:00 to  16:00  uses Room14
tut312d        from   Fri.   12:00 to  14:00  uses Room14
tut312e        from   Fri.   10:00 to  12:00  uses Room14
tut312f        from   Fri.   8:00  to  10:00  uses Room14
tut312g        from   Thu.   17:00 to  19:00  uses Room14
tut312h        from   Thu.   15:00 to  17:00  uses Room14
tut312i        from   Thu.   13:00 to  15:00  uses Room14
tut312j        from   Thu.   11:00 to  13:00  uses Room14
tut312k        from   Thu.   9:00  to  11:00  uses Room14
tut312l        from   Wed.   18:00 to  20:00  uses Room14
tut312m        from   Wed.   16:00 to  18:00  uses Room14
tut312n        from   Wed.   14:00 to  16:00  uses Room14
tut312o        from   Wed.   10:00 to  12:00  uses Room14
tut312p        from   Thu.   17:00 to  19:00  uses Room13
tut312q        from   Thu.   15:00 to  17:00  uses Room13
tut315a        from   Fri.   16:00 to  18:00  uses Room13
tut315b        from   Fri.   14:00 to  16:00  uses Room13
tut315c        from   Fri.   12:00 to  14:00  uses Room13
tut315d        from   Fri.   10:00 to  12:00  uses Room13
tut315e        from   Fri.   8:00  to  10:00  uses Room13
tut307a        from   Wed.   8:00  to  10:00  uses Room14
tut307b        from   Tue.   17:00 to  19:00  uses Room14
tut307c        from   Tue.   15:00 to  17:00  uses Room14
tut307d        from   Thu.   13:00 to  15:00  uses Room13
tut317a        from   Thu.   11:00 to  13:00  uses Room13
tut317b        from   Thu.   9:00  to  11:00  uses Room13
tut317c        from   Wed.   18:00 to  20:00  uses Room13
tut317d        from   Wed.   16:00 to  18:00  uses Room13
tut317e        from   Wed.   14:00 to  16:00  uses Room13
tut317f        from   Wed.   10:00 to  12:00  uses Room13
```

```
tut317g        from   Wed.   8:00  to  10:00  uses Room13
tut317h        from   Tue.   17:00 to  19:00  uses Room13
tut317i        from   Tue.   15:00 to  17:00  uses Room13
tut317j        from   Tue.   13:00 to  15:00  uses Room13
tut317k        from   Tue.   11:00 to  13:00  uses Room13
tut317l        from   Tue.   9:00  to  11:00  uses Room13
tut317m        from   Mon.   18:00 to  20:00  uses Room13
tut111a        from   Mon.   16:00 to  18:00  uses Room13
tut111b        from   Mon.   14:00 to  16:00  uses Room13
tut111c        from   Mon.   12:00 to  14:00  uses Room13
tut111d        from   Mon.   10:00 to  12:00  uses Room13
tut111e        from   Mon.   8:00  to  10:00  uses Room13
tut111f        from   Fri.   16:00 to  18:00  uses Room12
tut111g        from   Fri.   14:00 to  16:00  uses Room12
tut111h        from   Fri.   12:00 to  14:00  uses Room12
tut211a        from   Mon.   18:00 to  20:00  uses Room15
tut211b        from   Mon.   16:00 to  18:00  uses Room15
tut211c        from   Mon.   14:00 to  16:00  uses Room15
tut211d        from   Mon.   12:00 to  14:00  uses Room15
tut211e        from   Mon.   10:00 to  12:00  uses Room15
tut211f        from   Fri.   10:00 to  12:00  uses Room12
tut212a        from   Tue.   13:00 to  15:00  uses Room14
tut212b        from   Tue.   11:00 to  13:00  uses Room14
tut212c        from   Tue.   9:00  to  11:00  uses Room14
tut212d        from   Mon.   18:00 to  20:00  uses Room14
tut212e        from   Mon.   16:00 to  18:00  uses Room14
tut212f        from   Mon.   14:00 to  16:00  uses Room14
tut212g        from   Mon.   12:00 to  14:00  uses Room14
tut212h        from   Mon.   10:00 to  12:00  uses Room14
tut212i        from   Fri.   8:00  to  10:00  uses Room12
tut301a        from   Fri.   16:00 to  18:00  uses Room15
tut301b        from   Fri.   14:00 to  16:00  uses Room15
tut302a        from   Thu.   17:00 to  19:00  uses Room12
tut302b        from   Thu.   15:00 to  17:00  uses Room12
tut303a        from   Thu.   13:00 to  15:00  uses Room12
tut303b        from   Thu.   11:00 to  13:00  uses Room12
tut303c        from   Thu.   9:00  to  11:00  uses Room12
tut303d        from   Wed.   18:00 to  20:00  uses Room12
tut304a        from   Mon.   8:00  to  10:00  uses Room15
tut305a        from   Mon.   8:00  to  10:00  uses Room14
tut305b        from   Wed.   16:00 to  18:00  uses Room12
tut306a        from   Wed.   14:00 to  16:00  uses Room12
tut308a        from   Mon.   9:00  to  10:00  uses Room12
```

```
tut308b        from   Mon.   8:00  to   9:00  uses Room12
tut308c        from   Mon.   8:00  to   9:00  uses Room11
tut401a        from   Wed.   10:00 to  12:00  uses Room12
tut401b        from   Wed.   8:00  to  10:00  uses Room12
tut401c        from   Tue.   17:00 to  19:00  uses Room12
tut401d        from   Tue.   15:00 to  17:00  uses Room12
tut401e        from   Tue.   13:00 to  15:00  uses Room12
tut401f        from   Tue.   11:00 to  13:00  uses Room12
tut401g        from   Tue.   9:00  to  11:00  uses Room12
tut401h        from   Mon.   18:00 to  20:00  uses Room12
tut401i        from   Mon.   16:00 to  18:00  uses Room12
tut401j        from   Mon.   14:00 to  16:00  uses Room12
tut401k        from   Mon.   12:00 to  14:00  uses Room12
tut401l        from   Mon.   10:00 to  12:00  uses Room12
tut401m        from   Fri.   16:00 to  18:00  uses Room11
tut401n        from   Fri.   14:00 to  16:00  uses Room11
tut401o        from   Fri.   12:00 to  14:00  uses Room11
tut402a        from   Fri.   10:00 to  12:00  uses Room11
tut402b        from   Fri.   8:00  to  10:00  uses Room11
tut402c        from   Thu.   17:00 to  19:00  uses Room11
tut402d        from   Thu.   15:00 to  17:00  uses Room11
tut402e        from   Thu.   13:00 to  15:00  uses Room11
tut402f        from   Thu.   11:00 to  13:00  uses Room11
tut402g        from   Thu.   9:00  to  11:00  uses Room11
tut402h        from   Wed.   18:00 to  20:00  uses Room11
tut402i        from   Wed.   16:00 to  18:00  uses Room11
tut402j        from   Wed.   14:00 to  16:00  uses Room11
tut403a        from   Wed.   11:00 to  13:00  uses Room11
tut403b        from   Wed.   9:00  to  11:00  uses Room11
tut403c        from   Tue.   18:00 to  20:00  uses Room11
tut403d        from   Tue.   16:00 to  18:00  uses Room11
tut403e        from   Tue.   14:00 to  16:00  uses Room11
tut404a        from   Tue.   12:00 to  14:00  uses Room11
tut404b        from   Tue.   10:00 to  12:00  uses Room11
tut404c        from   Tue.   8:00  to  10:00  uses Room11
tut404d        from   Mon.   17:00 to  19:00  uses Room11
tut404e        from   Mon.   15:00 to  17:00  uses Room11
tut406a        from   Mon.   13:00 to  15:00  uses Room11
tut406b        from   Mon.   11:00 to  13:00  uses Room11
tut406c        from   Mon.   9:00  to  11:00  uses Room11
```

Appendix 4 - Detail timetable using IloSetTimeForward goal

```
----------------------TIMETABLE SOLUTION-------------------------

sub102          from   Mon.   14:00 to  16:00  uses Room1A
sub110A         from   Wed.   8:00  to  10:00  uses Room4A
sub110B         from   Tue.   13:00 to  15:00  uses Room4A
sub201          from   Mon.   14:00 to  16:00  uses Room2A
sub214A         from   Thu.   8:00  to  10:00  uses Room1A
sub214B         from   Mon.   8:00  to  10:00  uses Room1A
sub311A         from   Tue.   10:00 to  12:00  uses Room2A
sub311B         from   Tue.   8:00  to  10:00  uses Room2A
sub312A         from   Mon.   16:00 to  18:00  uses Room3A
sub312B         from   Mon.   10:00 to  12:00  uses Room3A
sub315          from   Mon.   12:00 to  14:00  uses Room1A
sub307          from   Wed.   8:00  to   9:00  uses Room1A
sub317A         from   Mon.   10:00 to  12:00  uses Room2A
sub317B         from   Mon.   8:00  to  10:00  uses Room2A
sub111A         from   Wed.   16:00 to  18:00  uses Room1A
sub111B         from   Wed.   9:00  to  11:00  uses Room1A
sub211A         from   Wed.   14:00 to  16:00  uses Room1A
sub211B         from   Wed.   11:00 to  13:00  uses Room1A
sub212A         from   Wed.   8:00  to  10:00  uses Room2A
sub212B         from   Tue.   15:00 to  17:00  uses Room2A
sub301          from   Mon.   10:00 to  12:00  uses Room1A
sub302          from   Tue.   16:00 to  18:00  uses Room1A
sub303          from   Tue.   15:00 to  16:00  uses Room1A
sub304          from   Tue.   12:00 to  13:00  uses Room1A
sub305          from   Tue.   11:00 to  12:00  uses Room1A
sub306          from   Tue.   9:00  to  11:00  uses Room1A
sub308          from   Tue.   8:00  to   9:00  uses Room1A
sub401A         from   Wed.   14:00 to  16:00  uses Room2A
sub401B         from   Wed.   10:00 to  12:00  uses Room2A
sub402          from   Mon.   8:00  to  10:00  uses Room3A
sub403          from   Mon.   12:00 to  14:00  uses Room2A
sub404          from   Tue.   13:00 to  15:00  uses Room1A
sub406          from   Mon.   16:00 to  18:00  uses Room1A
tut102a         from   Tue.   13:00 to  15:00  uses Room17
```

```
tut102b        from   Tue.   11:00 to  13:00  uses Room17
tut102c        from   Tue.   9:00  to  11:00  uses Room17
tut102d        from   Mon.   18:00 to  20:00  uses Room17
tut102e        from   Mon.   16:00 to  18:00  uses Room17
tut110a        from   Tue.   9:00  to  11:00  uses Room18
tut110b        from   Mon.   18:00 to  20:00  uses Room18
tut110c        from   Mon.   16:00 to  18:00  uses Room18
tut110d        from   Mon.   14:00 to  16:00  uses Room18
tut110e        from   Mon.   12:00 to  14:00  uses Room18
tut110f        from   Mon.   10:00 to  12:00  uses Room18
tut110g        from   Mon.   8:00  to  10:00  uses Room18
tut110h        from   Fri.   16:00 to  18:00  uses Room16
tut110i        from   Fri.   14:00 to  16:00  uses Room16
tut110j        from   Fri.   12:00 to  14:00  uses Room16
tut110k        from   Fri.   10:00 to  12:00  uses Room16
tut110l        from   Fri.   8:00  to  10:00  uses Room16
tut110m        from   Tue.   15:00 to  17:00  uses Room16
tut110n        from   Tue.   11:00 to  13:00  uses Room16
tut110o        from   Mon.   8:00  to  10:00  uses Room16
tut201a        from   Thu.   17:00 to  19:00  uses Room16
tut201b        from   Thu.   15:00 to  17:00  uses Room16
tut201c        from   Thu.   13:00 to  15:00  uses Room16
tut201d        from   Thu.   11:00 to  13:00  uses Room16
tut201e        from   Thu.   9:00  to  11:00  uses Room16
tut201f        from   Wed.   18:00 to  20:00  uses Room16
tut201g        from   Wed.   16:00 to  18:00  uses Room16
tut201h        from   Wed.   14:00 to  16:00  uses Room16
tut214a        from   Wed.   10:00 to  12:00  uses Room16
tut214b        from   Wed.   8:00  to  10:00  uses Room16
tut214c        from   Tue.   17:00 to  19:00  uses Room16
tut214d        from   Tue.   13:00 to  15:00  uses Room16
tut214e        from   Tue.   9:00  to  11:00  uses Room16
tut214f        from   Mon.   18:00 to  20:00  uses Room16
tut214g        from   Mon.   16:00 to  18:00  uses Room16
tut214h        from   Mon.   14:00 to  16:00  uses Room16
tut214i        from   Mon.   12:00 to  14:00  uses Room16
tut214j        from   Mon.   10:00 to  12:00  uses Room16
tut214k        from   Fri.   16:00 to  18:00  uses Room15
tut214m        from   Fri.   14:00 to  16:00  uses Room15
tut311a        from   Fri.   12:00 to  14:00  uses Room15
tut311b        from   Fri.   10:00 to  12:00  uses Room15
tut311c        from   Fri.   8:00  to  10:00  uses Room15
tut311d        from   Thu.   17:00 to  19:00  uses Room15
```

```
tut311e        from   Thu.   15:00 to  17:00  uses Room15
tut311f        from   Thu.   13:00 to  15:00  uses Room15
tut311g        from   Thu.   11:00 to  13:00  uses Room15
tut311h        from   Thu.   9:00  to  11:00  uses Room15
tut311i        from   Wed.   18:00 to  20:00  uses Room15
tut311j        from   Wed.   16:00 to  18:00  uses Room15
tut311k        from   Wed.   14:00 to  16:00  uses Room15
tut311l        from   Wed.   10:00 to  12:00  uses Room15
tut311m        from   Wed.   8:00  to  10:00  uses Room15
tut311n        from   Tue.   17:00 to  19:00  uses Room15
tut311o        from   Tue.   15:00 to  17:00  uses Room15
tut312a        from   Tue.   13:00 to  15:00  uses Room15
tut312b        from   Fri.   16:00 to  18:00  uses Room14
tut312c        from   Fri.   14:00 to  16:00  uses Room14
tut312d        from   Fri.   12:00 to  14:00  uses Room14
tut312e        from   Fri.   10:00 to  12:00  uses Room14
tut312f        from   Fri.   8:00  to  10:00  uses Room14
tut312g        from   Thu.   17:00 to  19:00  uses Room14
tut312h        from   Thu.   15:00 to  17:00  uses Room14
tut312i        from   Thu.   13:00 to  15:00  uses Room14
tut312j        from   Thu.   11:00 to  13:00  uses Room14
tut312k        from   Thu.   9:00  to  11:00  uses Room14
tut312l        from   Wed.   18:00 to  20:00  uses Room14
tut312m        from   Wed.   16:00 to  18:00  uses Room14
tut312n        from   Wed.   14:00 to  16:00  uses Room14
tut312o        from   Wed.   8:00  to  10:00  uses Room14
tut312p        from   Fri.   16:00 to  18:00  uses Room13
tut312q        from   Fri.   14:00 to  16:00  uses Room13
tut315a        from   Fri.   12:00 to  14:00  uses Room13
tut315b        from   Fri.   10:00 to  12:00  uses Room13
tut315c        from   Fri.   8:00  to  10:00  uses Room13
tut315d        from   Thu.   17:00 to  19:00  uses Room13
tut315e        from   Thu.   15:00 to  17:00  uses Room13
tut307a        from   Wed.   10:00 to  12:00  uses Room14
tut307b        from   Tue.   17:00 to  19:00  uses Room14
tut307c        from   Tue.   15:00 to  17:00  uses Room14
tut307d        from   Thu.   13:00 to  15:00  uses Room13
tut317a        from   Thu.   11:00 to  13:00  uses Room13
tut317b        from   Thu.   9:00  to  11:00  uses Room13
tut317c        from   Wed.   18:00 to  20:00  uses Room13
tut317d        from   Wed.   16:00 to  18:00  uses Room13
tut317e        from   Wed.   14:00 to  16:00  uses Room13
tut317f        from   Wed.   10:00 to  12:00  uses Room13
```

```
tut317g        from   Wed.   8:00  to  10:00  uses Room13
tut317h        from   Tue.   17:00 to  19:00  uses Room13
tut317i        from   Tue.   15:00 to  17:00  uses Room13
tut317j        from   Tue.   13:00 to  15:00  uses Room13
tut317k        from   Tue.   11:00 to  13:00  uses Room13
tut317l        from   Tue.   9:00  to  11:00  uses Room13
tut317m        from   Mon.   18:00 to  20:00  uses Room13
tut111a        from   Mon.   16:00 to  18:00  uses Room13
tut111b        from   Mon.   14:00 to  16:00  uses Room13
tut111c        from   Mon.   12:00 to  14:00  uses Room13
tut111d        from   Mon.   10:00 to  12:00  uses Room13
tut111e        from   Mon.   8:00  to  10:00  uses Room13
tut111f        from   Fri.   16:00 to  18:00  uses Room12
tut111g        from   Fri.   14:00 to  16:00  uses Room12
tut111h        from   Fri.   12:00 to  14:00  uses Room12
tut211a        from   Tue.   11:00 to  13:00  uses Room15
tut211b        from   Tue.   9:00  to  11:00  uses Room15
tut211c        from   Mon.   18:00 to  20:00  uses Room15
tut211d        from   Mon.   16:00 to  18:00  uses Room15
tut211e        from   Mon.   10:00 to  12:00  uses Room15
tut211f        from   Fri.   10:00 to  12:00  uses Room12
tut212a        from   Tue.   13:00 to  15:00  uses Room14
tut212b        from   Tue.   11:00 to  13:00  uses Room14
tut212c        from   Tue.   9:00  to  11:00  uses Room14
tut212d        from   Mon.   18:00 to  20:00  uses Room14
tut212e        from   Mon.   16:00 to  18:00  uses Room14
tut212f        from   Mon.   14:00 to  16:00  uses Room14
tut212g        from   Mon.   12:00 to  14:00  uses Room14
tut212h        from   Mon.   10:00 to  12:00  uses Room14
tut212i        from   Fri.   8:00  to  10:00  uses Room12
tut301a        from   Mon.   14:00 to  16:00  uses Room15
tut301b        from   Mon.   12:00 to  14:00  uses Room15
tut302a        from   Thu.   17:00 to  19:00  uses Room12
tut302b        from   Thu.   15:00 to  17:00  uses Room12
tut303a        from   Thu.   13:00 to  15:00  uses Room12
tut303b        from   Thu.   11:00 to  13:00  uses Room12
tut303c        from   Thu.   9:00  to  11:00  uses Room12
tut303d        from   Wed.   18:00 to  20:00  uses Room12
tut304a        from   Mon.   8:00  to  10:00  uses Room15
tut305a        from   Mon.   8:00  to  10:00  uses Room14
tut305b        from   Wed.   16:00 to  18:00  uses Room12
tut306a        from   Wed.   14:00 to  16:00  uses Room12
tut308a        from   Wed.   11:00 to  12:00  uses Room12
```

```
tut308b        from   Wed.   10:00 to  11:00  uses Room12
tut308c        from   Wed.   12:00 to  13:00  uses Room11
tut401a        from   Wed.   8:00  to  10:00  uses Room12
tut401b        from   Tue.   17:00 to  19:00  uses Room12
tut401c        from   Tue.   15:00 to  17:00  uses Room12
tut401d        from   Tue.   13:00 to  15:00  uses Room12
tut401e        from   Tue.   11:00 to  13:00  uses Room12
tut401f        from   Tue.   9:00  to  11:00  uses Room12
tut401g        from   Mon.   18:00 to  20:00  uses Room12
tut401h        from   Mon.   16:00 to  18:00  uses Room12
tut401i        from   Mon.   14:00 to  16:00  uses Room12
tut401j        from   Mon.   12:00 to  14:00  uses Room12
tut401k        from   Mon.   10:00 to  12:00  uses Room12
tut401l        from   Mon.   8:00  to  10:00  uses Room12
tut401m        from   Fri.   16:00 to  18:00  uses Room11
tut401n        from   Fri.   14:00 to  16:00  uses Room11
tut401o        from   Fri.   12:00 to  14:00  uses Room11
tut402a        from   Fri.   10:00 to  12:00  uses Room11
tut402b        from   Fri.   8:00  to  10:00  uses Room11
tut402c        from   Thu.   17:00 to  19:00  uses Room11
tut402d        from   Thu.   15:00 to  17:00  uses Room11
tut402e        from   Thu.   13:00 to  15:00  uses Room11
tut402f        from   Thu.   11:00 to  13:00  uses Room11
tut402g        from   Thu.   9:00  to  11:00  uses Room11
tut402h        from   Wed.   18:00 to  20:00  uses Room11
tut402i        from   Wed.   16:00 to  18:00  uses Room11
tut402j        from   Wed.   14:00 to  16:00  uses Room11
tut403a        from   Wed.   10:00 to  12:00  uses Room11
tut403b        from   Wed.   8:00  to  10:00  uses Room11
tut403c        from   Tue.   17:00 to  19:00  uses Room11
tut403d        from   Tue.   15:00 to  17:00  uses Room11
tut403e        from   Tue.   13:00 to  15:00  uses Room11
tut404a        from   Tue.   11:00 to  13:00  uses Room11
tut404b        from   Tue.   9:00  to  11:00  uses Room11
tut404c        from   Mon.   18:00 to  20:00  uses Room11
tut404d        from   Mon.   16:00 to  18:00  uses Room11
tut404e        from   Mon.   14:00 to  16:00  uses Room11
tut406a        from   Mon.   12:00 to  14:00  uses Room11
tut406b        from   Mon.   10:00 to  12:00  uses Room11
tut406c        from   Mon.   8:00  to  10:00  uses Room11
```

Appendix 5 - Detail timetable using IloSetTimeBackward goal

```
----------------------TIMETABLE SOLUTION-------------------------

sub102          from   Tue.   16:00 to  18:00  uses Room1A
sub110A         from   Thu.   9:00  to  11:00  uses Room4A
sub110B         from   Thu.   13:00 to  15:00  uses Room4A
sub201          from   Thu.   11:00 to  13:00  uses Room2A
sub214A         from   Fri.   16:00 to  18:00  uses Room1A
sub214B         from   Tue.   10:00 to  12:00  uses Room1A
sub311A         from   Thu.   14:00 to  16:00  uses Room2A
sub311B         from   Thu.   16:00 to  18:00  uses Room2A
sub312A         from   Fri.   14:00 to  16:00  uses Room3A
sub312B         from   Fri.   16:00 to  18:00  uses Room3A
sub315          from   Thu.   8:00  to  10:00  uses Room1A
sub307          from   Wed.   17:00 to  18:00  uses Room1A
sub317A         from   Mon.   11:00 to  13:00  uses Room2A
sub317B         from   Mon.   13:00 to  15:00  uses Room2A
sub111A         from   Thu.   10:00 to  12:00  uses Room1A
sub111B         from   Wed.   15:00 to  17:00  uses Room1A
sub211A         from   Wed.   9:00  to  11:00  uses Room1A
sub211B         from   Wed.   11:00 to  13:00  uses Room1A
sub212A         from   Wed.   16:00 to  18:00  uses Room2A
sub212B         from   Fri.   12:00 to  14:00  uses Room2A
sub301          from   Thu.   13:00 to  15:00  uses Room1A
sub302          from   Thu.   15:00 to  17:00  uses Room1A
sub303          from   Thu.   17:00 to  18:00  uses Room1A
sub304          from   Thu.   12:00 to  13:00  uses Room1A
sub305          from   Fri.   8:00  to   9:00  uses Room1A
sub306          from   Fri.   11:00 to  13:00  uses Room1A
sub308          from   Fri.   13:00 to  14:00  uses Room1A
sub401A         from   Wed.   11:00 to  13:00  uses Room2A
sub401B         from   Wed.   14:00 to  16:00  uses Room2A
sub402          from   Fri.   12:00 to  14:00  uses Room3A
sub403          from   Fri.   16:00 to  18:00  uses Room2A
sub404          from   Fri.   14:00 to  16:00  uses Room1A
sub406          from   Fri.   9:00  to  11:00  uses Room1A
tut102a         from   Fri.   8:00  to  10:00  uses Room17
tut102b         from   Fri.   10:00 to  12:00  uses Room17
```

```
tut102c        from   Fri.   12:00 to  14:00  uses Room17
tut102d        from   Fri.   14:00 to  16:00  uses Room17
tut102e        from   Fri.   16:00 to  18:00  uses Room17
tut110a        from   Thu.   15:00 to  17:00  uses Room18
tut110b        from   Thu.   17:00 to  19:00  uses Room18
tut110c        from   Fri.   8:00  to  10:00  uses Room18
tut110d        from   Fri.   10:00 to  12:00  uses Room18
tut110e        from   Fri.   12:00 to  14:00  uses Room18
tut110f        from   Fri.   14:00 to  16:00  uses Room18
tut110g        from   Fri.   16:00 to  18:00  uses Room18
tut110h        from   Mon.   9:00  to  11:00  uses Room16
tut110i        from   Mon.   11:00 to  13:00  uses Room16
tut110j        from   Mon.   13:00 to  15:00  uses Room16
tut110k        from   Mon.   15:00 to  17:00  uses Room16
tut110l        from   Mon.   17:00 to  19:00  uses Room16
tut110m        from   Tue.   8:00  to  10:00  uses Room16
tut110n        from   Tue.   10:00 to  12:00  uses Room16
tut110o        from   Thu.   11:00 to  13:00  uses Room16
tut201a        from   Thu.   13:00 to  15:00  uses Room16
tut201b        from   Thu.   15:00 to  17:00  uses Room16
tut201c        from   Thu.   17:00 to  19:00  uses Room16
tut201d        from   Fri.   8:00  to  10:00  uses Room16
tut201e        from   Fri.   10:00 to  12:00  uses Room16
tut201f        from   Fri.   12:00 to  14:00  uses Room16
tut201g        from   Fri.   14:00 to  16:00  uses Room16
tut201h        from   Fri.   16:00 to  18:00  uses Room16
tut214a        from   Tue.   12:00 to  14:00  uses Room16
tut214b        from   Tue.   14:00 to  16:00  uses Room16
tut214c        from   Tue.   16:00 to  18:00  uses Room16
tut214d        from   Tue.   18:00 to  20:00  uses Room16
tut214e        from   Wed.   9:00  to  11:00  uses Room16
tut214f        from   Wed.   11:00 to  13:00  uses Room16
tut214g        from   Wed.   14:00 to  16:00  uses Room16
tut214h        from   Wed.   16:00 to  18:00  uses Room16
tut214i        from   Wed.   18:00 to  20:00  uses Room16
tut214j        from   Thu.   9:00  to  11:00  uses Room16
tut214k        from   Mon.   9:00  to  11:00  uses Room15
tut214m        from   Mon.   11:00 to  13:00  uses Room15
tut311a        from   Mon.   13:00 to  15:00  uses Room15
tut311b        from   Mon.   15:00 to  17:00  uses Room15
tut311c        from   Mon.   17:00 to  19:00  uses Room15
tut311d        from   Tue.   8:00  to  10:00  uses Room15
tut311e        from   Tue.   10:00 to  12:00  uses Room15
```

```
tut311f        from   Tue.   12:00 to  14:00  uses Room15
tut311g        from   Tue.   14:00 to  16:00  uses Room15
tut311h        from   Tue.   16:00 to  18:00  uses Room15
tut311i        from   Tue.   18:00 to  20:00  uses Room15
tut311j        from   Wed.   9:00  to  11:00  uses Room15
tut311k        from   Wed.   11:00 to  13:00  uses Room15
tut311l        from   Wed.   14:00 to  16:00  uses Room15
tut311m        from   Wed.   16:00 to  18:00  uses Room15
tut311n        from   Fri.   8:00  to  10:00  uses Room15
tut311o        from   Fri.   10:00 to  12:00  uses Room15
tut312a        from   Wed.   18:00 to  20:00  uses Room15
tut312b        from   Mon.   9:00  to  11:00  uses Room14
tut312c        from   Mon.   11:00 to  13:00  uses Room14
tut312d        from   Mon.   13:00 to  15:00  uses Room14
tut312e        from   Mon.   15:00 to  17:00  uses Room14
tut312f        from   Mon.   17:00 to  19:00  uses Room14
tut312g        from   Tue.   8:00  to  10:00  uses Room14
tut312h        from   Tue.   10:00 to  12:00  uses Room14
tut312i        from   Tue.   12:00 to  14:00  uses Room14
tut312j        from   Tue.   14:00 to  16:00  uses Room14
tut312k        from   Tue.   16:00 to  18:00  uses Room14
tut312l        from   Tue.   18:00 to  20:00  uses Room14
tut312m        from   Wed.   9:00  to  11:00  uses Room14
tut312n        from   Wed.   11:00 to  13:00  uses Room14
tut312o        from   Wed.   16:00 to  18:00  uses Room14
tut312p        from   Mon.   9:00  to  11:00  uses Room13
tut312q        from   Mon.   11:00 to  13:00  uses Room13
tut315a        from   Fri.   8:00  to  10:00  uses Room13
tut315b        from   Fri.   10:00 to  12:00  uses Room13
tut315c        from   Fri.   12:00 to  14:00  uses Room13
tut315d        from   Fri.   14:00 to  16:00  uses Room13
tut315e        from   Fri.   16:00 to  18:00  uses Room13
tut307a        from   Wed.   14:00 to  16:00  uses Room14
tut307b        from   Wed.   18:00 to  20:00  uses Room14
tut307c        from   Fri.   12:00 to  14:00  uses Room14
tut307d        from   Mon.   13:00 to  15:00  uses Room13
tut317a        from   Mon.   15:00 to  17:00  uses Room13
tut317b        from   Mon.   17:00 to  19:00  uses Room13
tut317c        from   Tue.   8:00  to  10:00  uses Room13
tut317d        from   Tue.   10:00 to  12:00  uses Room13
tut317e        from   Tue.   12:00 to  14:00  uses Room13
tut317f        from   Tue.   14:00 to  16:00  uses Room13
tut317g        from   Tue.   16:00 to  18:00  uses Room13
```

```
tut317h        from   Tue.   18:00 to  20:00  uses Room13
tut317i        from   Wed.   9:00  to  11:00  uses Room13
tut317j        from   Wed.   11:00 to  13:00  uses Room13
tut317k        from   Wed.   14:00 to  16:00  uses Room13
tut317l        from   Wed.   16:00 to  18:00  uses Room13
tut317m        from   Wed.   18:00 to  20:00  uses Room13
tut111a        from   Thu.   9:00  to  11:00  uses Room13
tut111b        from   Thu.   11:00 to  13:00  uses Room13
tut111c        from   Thu.   13:00 to  15:00  uses Room13
tut111d        from   Thu.   15:00 to  17:00  uses Room13
tut111e        from   Thu.   17:00 to  19:00  uses Room13
tut111f        from   Mon.   9:00  to  11:00  uses Room12
tut111g        from   Mon.   11:00 to  13:00  uses Room12
tut111h        from   Mon.   13:00 to  15:00  uses Room12
tut211a        from   Thu.   9:00  to  11:00  uses Room15
tut211b        from   Thu.   11:00 to  13:00  uses Room15
tut211c        from   Thu.   13:00 to  15:00  uses Room15
tut211d        from   Thu.   15:00 to  17:00  uses Room15
tut211e        from   Thu.   17:00 to  19:00  uses Room15
tut211f        from   Mon.   15:00 to  17:00  uses Room12
tut212a        from   Thu.   9:00  to  11:00  uses Room14
tut212b        from   Thu.   11:00 to  13:00  uses Room14
tut212c        from   Thu.   13:00 to  15:00  uses Room14
tut212d        from   Thu.   15:00 to  17:00  uses Room14
tut212e        from   Thu.   17:00 to  19:00  uses Room14
tut212f        from   Fri.   8:00  to  10:00  uses Room14
tut212g        from   Fri.   10:00 to  12:00  uses Room14
tut212h        from   Fri.   14:00 to  16:00  uses Room14
tut212i        from   Mon.   17:00 to  19:00  uses Room12
tut301a        from   Fri.   14:00 to  16:00  uses Room15
tut301b        from   Fri.   16:00 to  18:00  uses Room15
tut302a        from   Tue.   8:00  to  10:00  uses Room12
tut302b        from   Tue.   10:00 to  12:00  uses Room12
tut303a        from   Tue.   12:00 to  14:00  uses Room12
tut303b        from   Tue.   14:00 to  16:00  uses Room12
tut303c        from   Tue.   16:00 to  18:00  uses Room12
tut303d        from   Tue.   18:00 to  20:00  uses Room12
tut304a        from   Fri.   12:00 to  14:00  uses Room15
tut305a        from   Fri.   16:00 to  18:00  uses Room14
tut305b        from   Wed.   9:00  to  11:00  uses Room12
tut306a        from   Wed.   11:00 to  13:00  uses Room12
tut308a        from   Wed.   14:00 to  15:00  uses Room12
tut308b        from   Wed.   15:00 to  16:00  uses Room12
```

```
tut308c        from   Mon.   8:00  to   9:00  uses Room11
tut401a        from   Wed.   16:00 to  18:00  uses Room12
tut401b        from   Wed.   18:00 to  20:00  uses Room12
tut401c        from   Thu.   9:00  to  11:00  uses Room12
tut401d        from   Thu.   11:00 to  13:00  uses Room12
tut401e        from   Thu.   13:00 to  15:00  uses Room12
tut401f        from   Thu.   15:00 to  17:00  uses Room12
tut401g        from   Thu.   17:00 to  19:00  uses Room12
tut401h        from   Fri.   8:00  to  10:00  uses Room12
tut401i        from   Fri.   10:00 to  12:00  uses Room12
tut401j        from   Fri.   12:00 to  14:00  uses Room12
tut401k        from   Fri.   14:00 to  16:00  uses Room12
tut401l        from   Fri.   16:00 to  18:00  uses Room12
tut401m        from   Mon.   9:00  to  11:00  uses Room11
tut401n        from   Mon.   11:00 to  13:00  uses Room11
tut401o        from   Mon.   13:00 to  15:00  uses Room11
tut402a        from   Mon.   15:00 to  17:00  uses Room11
tut402b        from   Mon.   17:00 to  19:00  uses Room11
tut402c        from   Tue.   8:00  to  10:00  uses Room11
tut402d        from   Tue.   10:00 to  12:00  uses Room11
tut402e        from   Tue.   12:00 to  14:00  uses Room11
tut402f        from   Tue.   14:00 to  16:00  uses Room11
tut402g        from   Tue.   16:00 to  18:00  uses Room11
tut402h        from   Tue.   18:00 to  20:00  uses Room11
tut402i        from   Wed.   9:00  to  11:00  uses Room11
tut402j        from   Wed.   11:00 to  13:00  uses Room11
tut403a        from   Wed.   14:00 to  16:00  uses Room11
tut403b        from   Wed.   16:00 to  18:00  uses Room11
tut403c        from   Wed.   18:00 to  20:00  uses Room11
tut403d        from   Thu.   9:00  to  11:00  uses Room11
tut403e        from   Thu.   11:00 to  13:00  uses Room11
tut404a        from   Thu.   13:00 to  15:00  uses Room11
tut404b        from   Thu.   15:00 to  17:00  uses Room11
tut404c        from   Thu.   17:00 to  19:00  uses Room11
tut404d        from   Fri.   8:00  to  10:00  uses Room11
tut404e        from   Fri.   10:00 to  12:00  uses Room11
tut406a        from   Fri.   12:00 to  14:00  uses Room11
tut406b        from   Fri.   14:00 to  16:00  uses Room11
tut406c        from   Fri.   16:00 to  18:00  uses Room11
```